NEXT.JS Documentation

By,

Ajay Sreenivas Prudhvi,

GOVE.

Find the official NextJs documentation here

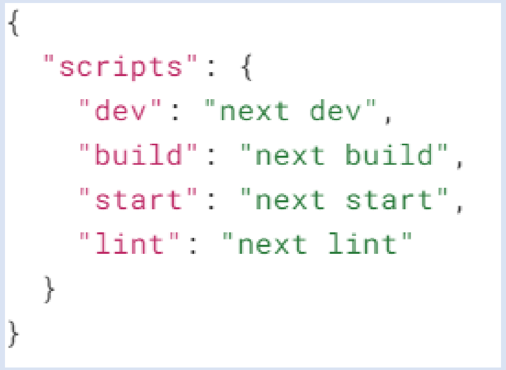
Getting started:

Installation :

Type this command in terminal for creating a Nextjs project

Command : npx create-next-app@latest <project_folder_name_here>

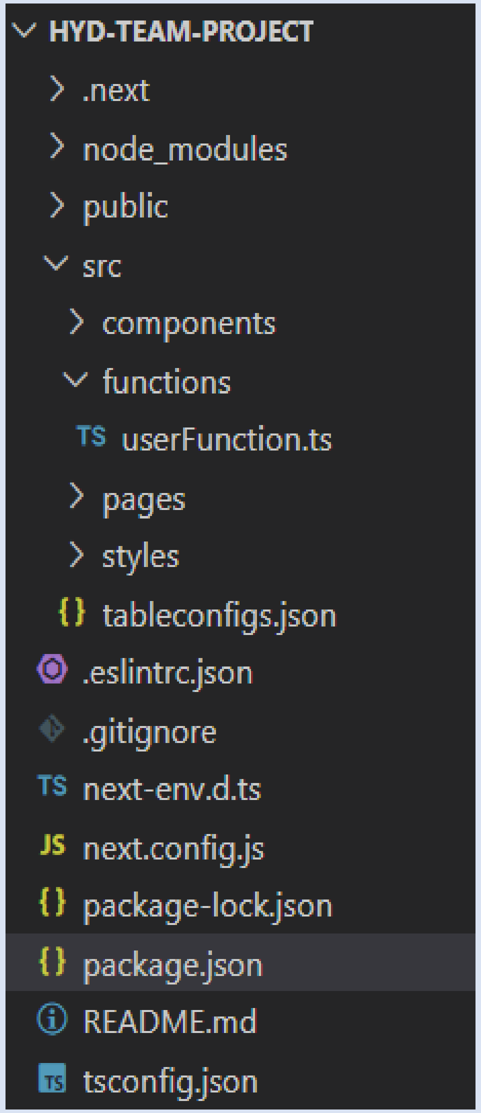
Make sure you have the following scripts in package.json file in the project folder



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

To run the project, type this command : **npm run dev**
and type this link in browser : http://localhost:3000

Essential folders/files to know:


```
HYD-TEAM-PROJECT
  .next
  node_modules
  public
  src
    components
    functions
      userFunction.ts
    pages
    styles
    tableconfigs.json
  .eslintrc.json
  .gitignore
  next-env.d.ts
  next.config.js
  package-lock.json
  package.json
  README.md
  tsconfig.json
```

package.json:

It is the file where you can find your project name, version dependencies(dependencies are external packages which are used in your project), scripts(commands that you can run in the terminal) etc.....

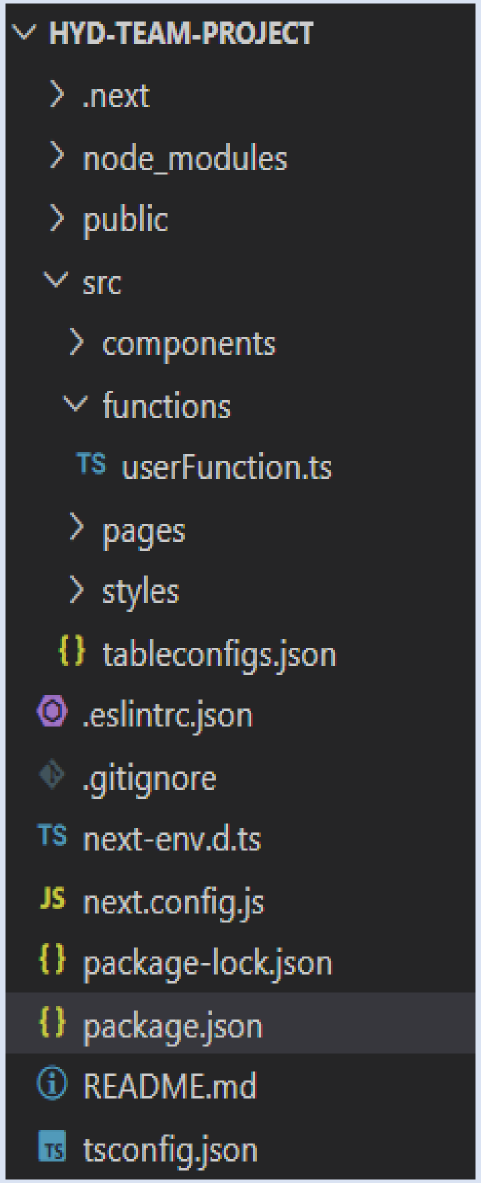
package-lock.json:

It has all the dependencies which are essential for your external packages.

.gitignore:

It has all files which are to be ignored while pushing into git repository.

Essential folders/files to know:



```
∨ HYD-TEAM-PROJECT
  > .next
  > node_modules
  > public
  ∨ src
    > components
    ∨ functions
      TS userFunction.ts
    > pages
    > styles
    {} tableconfigs.json
  .eslintrc.json
  .gitignore
  TS next-env.d.ts
  JS next.config.js
  {} package-lock.json
  {} package.json
  README.md
  tsconfig.json
```

node_modules:

You can find all the dependencies of your project with their code here.

public:

You can drop all the images you want to render here and access them anywhere

src:

In this project src folder is looking like the one as shown in the left picture. This is where you will be coding.

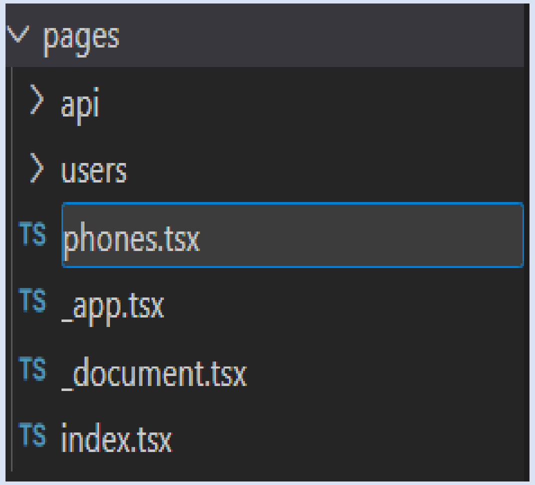
By default, it has pages and styles folders.

Pages folder has all the routes. We will discuss about this later in detail.

You can have all your custom css files in style files in styles folder with .css extension and use it in any route in pages.

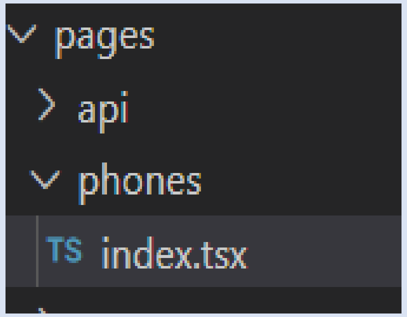
File System based routing:

- If you are from React, you may know about React router....In react we specify all the routes(routing is nothing but page linking and navigating) separately in a file and if the routes are very complex then it is very painful to specify every route and link them...If you are not familiar about this don't worry follow this and you will get good with routing in Next Js.
- We do routing in Next Js with the help of pages folder. We just need to add a file and write some code in it to display something, and you are done. Suppose I need to add a route "**/phones**" , I should create a file called **phones.tsx** in pages folder that's it... our phones route is ready.. If you want to see this, you should type the following URL in the browser : http://localhost:3000/phones


```
pages
  api
  users
  phones.tsx
  _app.tsx
  _document.tsx
  index.tsx
```

- If I want to add a route called "**/desktops**", I need to create a file called desktops.tsx in pages folder and type the following link in browser:
- http://localhost:3000/desktops

- We can implement the same by doing the following:



- The url remains the same just the folder structure changed.

Linking and Navigating:

It is used for connecting pages to pages. It can be done easily using link component in Next Js.
To use Link component in Next Js, we should import it:

Import statement:
Import Link from 'next/Link'

<Link> tag in Next Js is like anchor tag(<a>) in html, where we provide **href** attribute.
Syntax:
<Link href="/foo"> clicking here will take you to foo page**</Link>**

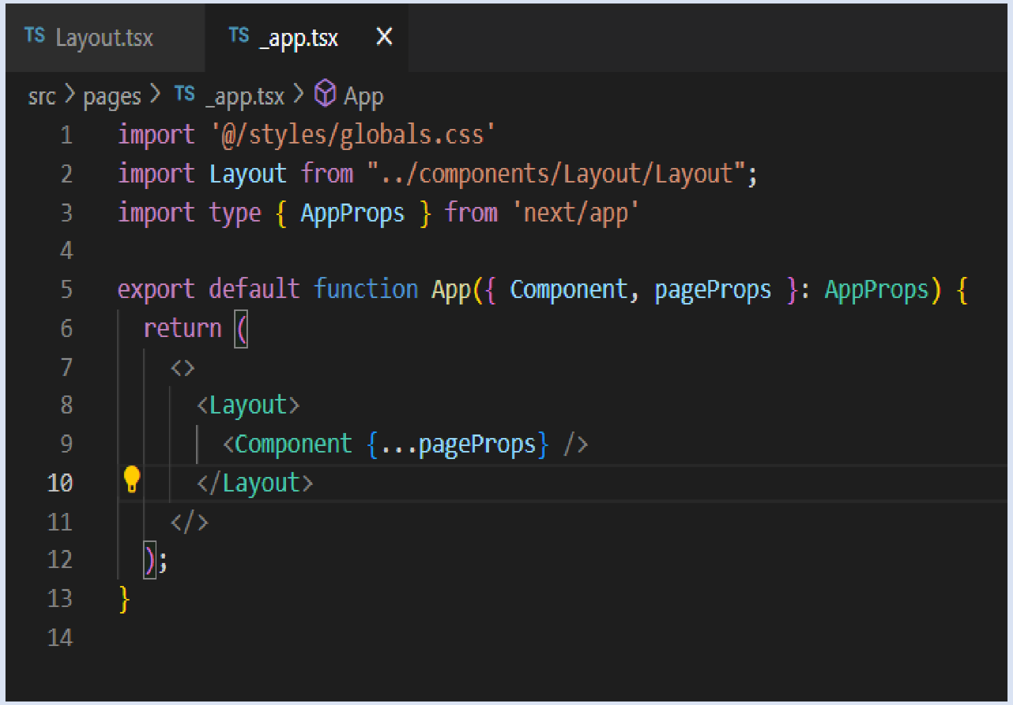
Layouts in NextJs

We need a layout for every website. Layout is nothing but header, footer, side bar, content of that page, etc...

Every time the page is loaded _app.tsx gets executed.. So, If we place any content in _app.tsx it will remain for every page...Thus, we can place our layout here...By doing this the layout is applied for all the pages.

_app.tsx looks like this, this is where specify the layout that is common to pages.

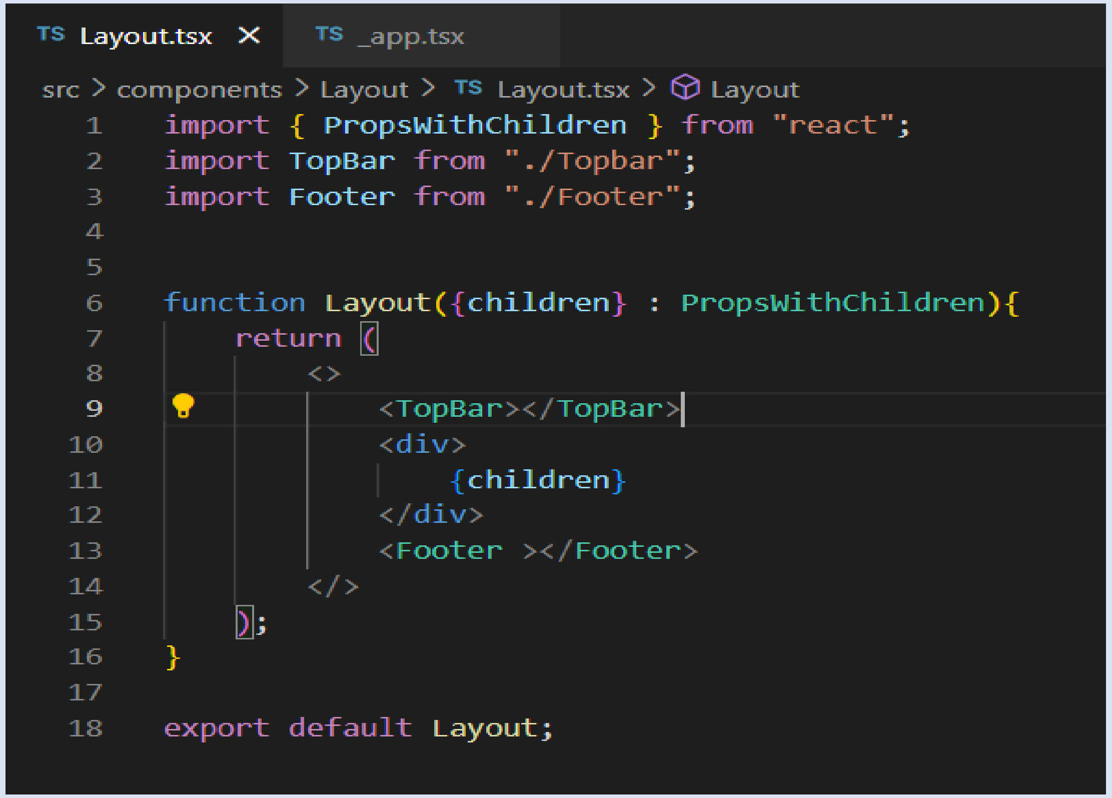
Layout looks like this



Layout.tsx | _app.tsx

src > pages > _app.tsx > App

```tsx
import '@/styles/globals.css'
import Layout from "../components/Layout/Layout";
import type { AppProps } from 'next/app'

export default function App({ Component, pageProps }: AppProps) {
  return (
    <>
      <Layout>
        <Component {...pageProps} />
      </Layout>
    </>
  );
}

```



Layout.tsx | _app.tsx

src > components > Layout > Layout.tsx > Layout

```tsx
import { PropsWithChildren } from "react";
import TopBar from "./Topbar";
import Footer from "./Footer";


function Layout({children} : PropsWithChildren){
    return (
        <>
            <TopBar></TopBar>
            <div>
                {children}
            </div>
            <Footer ></Footer>
        </>
    );
}

export default Layout;
```

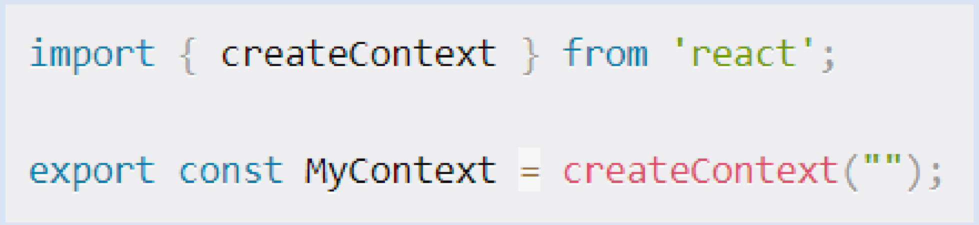
Providers in Next Js:

In NextJs, providers are typically used to manage global state or to provide context to components across your application(making data available to components without passing props, by this we can avoid prop drilling issue).

Need of useContext API:

- We can use React's built-in useContext API to create a provider for our application. We can use this API to manage global state and share data across components.

1) First, we need to create a context object.



```
import { createContext } from 'react';

export const MyContext = createContext("");
```

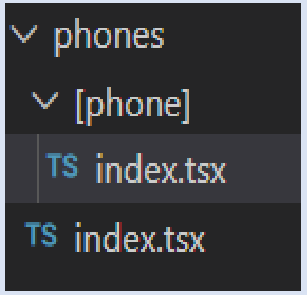
2) Wrap the components in a provider.

- Once the context object is created, we need to now wrap the components that need access to the shared data with a provider component. The provider component accepts a value prop that holds the shared data, and any component that is a child of the provider component can access that shared data.

- We can make href dynamic using template literals: href = { `/posts/${slug}`}

<u>Dynamic Routing:</u>

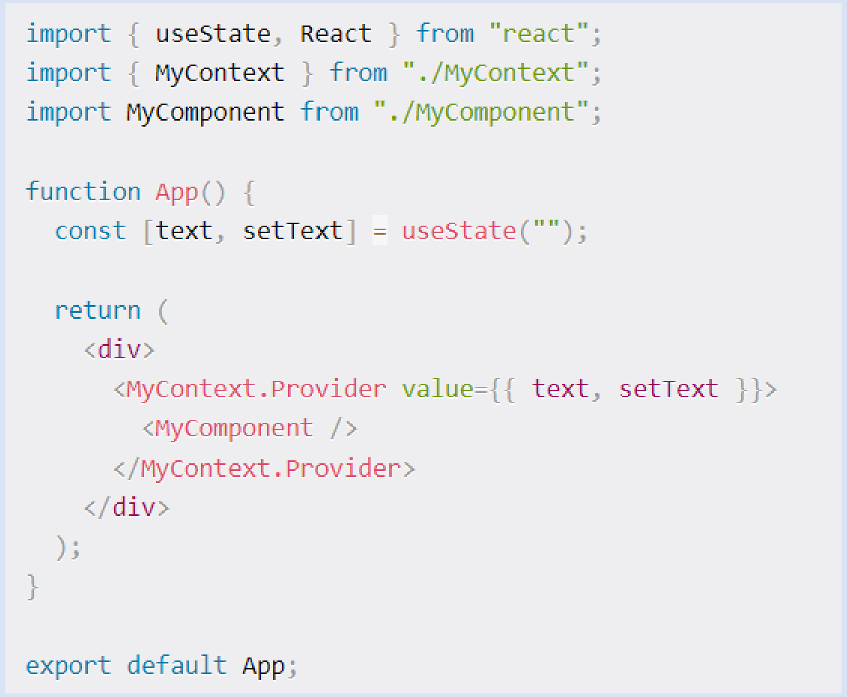
- When we don't know the URL ahead of time, we can use dynamic routing.
- Let us take an example of phones, say we are working on ecommerce application and when user clicks on a particular phone, it should show the page of that phone.
- Now we can't create a separate page for every phone right...it is painful, time consuming and the code is repetitive.. To shorten it we can use dynamic routing. We can make dynamic routes by specifying the folder name in square brackets ([folder name]) as shown below.



```
phones
  [phone]
    index.tsx
  index.tsx
```

- Along with this, we can use useRouter hook to get control of the route parameters

Like the path, path parameters, query parameters etc....
To know more about the methods available in useRouter hook follow this [link]



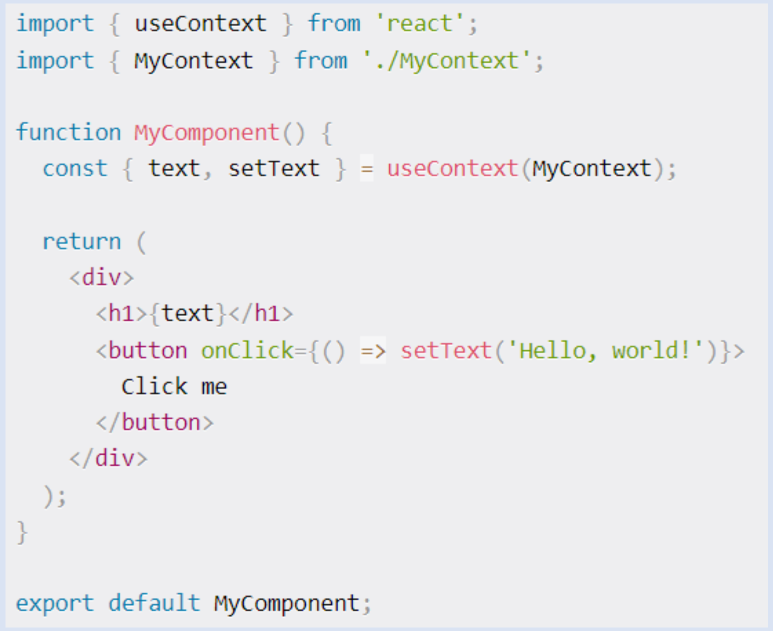
```
import { useState, React } from "react";
import { MyContext } from "./MyContext";
import MyComponent from "./MyComponent";

function App() {
  const [text, setText] = useState("");

  return (
    <div>
      <MyContext.Provider value={{ text, setText }}>
        <MyComponent />
      </MyContext.Provider>
    </div>
  );
}

export default App;
```

- In this code snippet, the variables text, setText can be accessed in the component MyComponent..notice that we haven't passed any props to MyComponent.. Still, we are able to access the variables...this avoids prop drilling.

3) Consume the context

- To consume the variables in the child component, we need to use useContext hook.

- To know more use cases for useContext hook, go through this blog



```
import { useContext } from 'react';
import { MyContext } from './MyContext';

function MyComponent() {
  const { text, setText } = useContext(MyContext);

  return (
    <div>
      <h1>{text}</h1>
      <button onClick={() => setText('Hello, world!')}>
        Click me
      </button>
    </div>
  );
}

export default MyComponent;
```

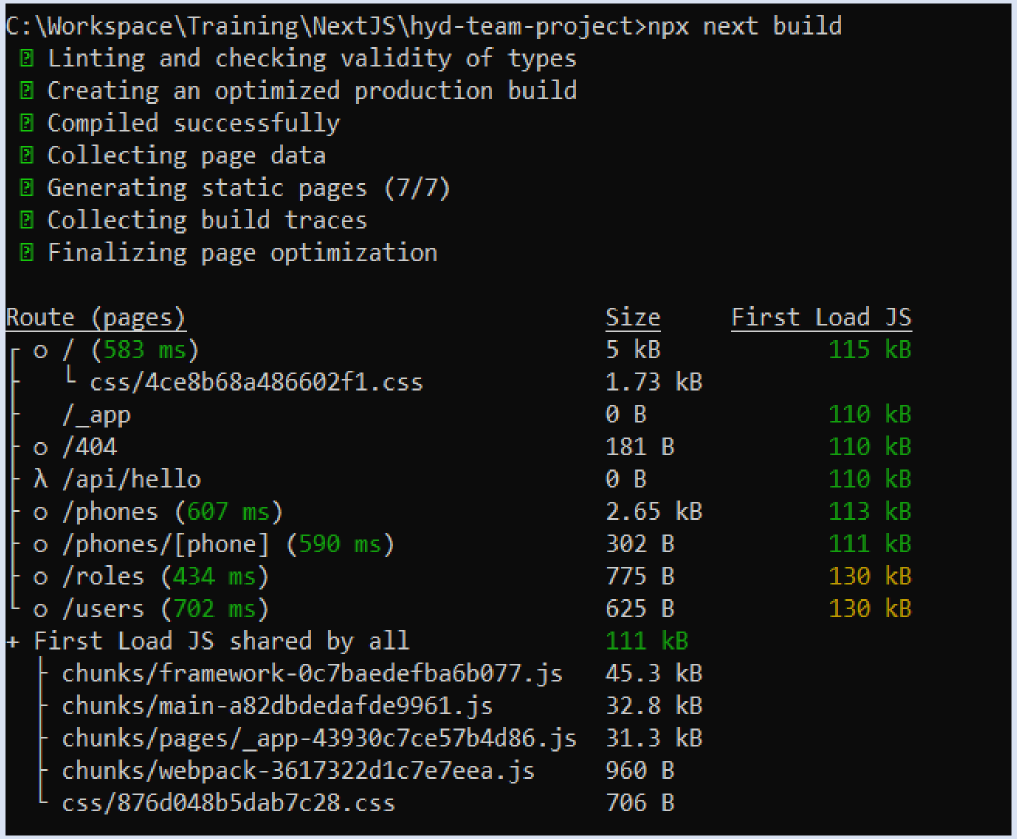
NextJs CLI:

- CLI refers to Command Line Interface, We can know about the commands we can use in Next JS through CLI. Find official NextJS CLI here
- To get the list of available commands type **npx next –h** in terminal.
- To start Next JS app type **npm run dev** in terminal. This will start application at http://localhost:300 by default. The default port can be changed by using the –p flag like this **npx next dev –p 4000 ,** this will start the application at http://localhost:4000

- **npx next build** command can be used to build the application in more optimized way.

Terminal output looks like this, it contains all the routes and sizes they occupy for loading etc..

- **npx next start** can be run only if it is compiled before using **npx next build**



```
C:\Workspace\Training\NextJS\hyd-team-project>npx next build
 ⍰ Linting and checking validity of types
 ⍰ Creating an optimized production build
 ⍰ Compiled successfully
 ⍰ Collecting page data
 ⍰ Generating static pages (7/7)
 ⍰ Collecting build traces
 ⍰ Finalizing page optimization

Route (pages)                              Size     First Load JS
┌ o / (583 ms)                             5 kB            115 kB
├   └ css/4ce8b68a486602f1.css             1.73 kB
├   /_app                                  0 B             110 kB
├ o /404                                   181 B           110 kB
├ λ /api/hello                             0 B             110 kB
├ o /phones (607 ms)                       2.65 kB         113 kB
├ o /phones/[phone] (590 ms)               302 B           111 kB
├ o /roles (434 ms)                        775 B           130 kB
└ o /users (702 ms)                        625 B           130 kB
+ First Load JS shared by all              111 kB
  ├ chunks/framework-0c7baedefba6b077.js   45.3 kB
  ├ chunks/main-a82dbdedafde9961.js        32.8 kB
  ├ chunks/pages/_app-43930c7ce57b4d86.js  31.3 kB
  ├ chunks/webpack-3617322d1c7e7eea.js     960 B
  └ css/876d048b5dab7c28.css               706 B
```

Material UI (MUI):

Material UI is a library of React components which can be customized. These are pre-built components which can be used in our code rather than building them from scratch again and again.

Official MUI documentation link : [follow this link]

Command for installing MUI: **npm install @mui/material @emotion/react @emotion/styled**

After clicking the link above you can see the menu bar on the left, you can find components section there...these are the react components we will be using in our code.

If we want to add custom css to these components, then we need to add **sx** attribute and specify the style we need in the opening tag like this : < Table **sx={{backgroundColor: "red"}}** > </Table>