Installing yarn : npm intall –global yarn

Creating nextjs app: yarn create next-app

Free Readme generator: readme.so

import Head from 'next/head'

<Head> - used as head tag in normal html

import Script from 'next/script'

<Scirpt> - we can lazyload or different option available (add google analytics etc…)

import Image from 'next/image'

<Image> - optimized, lazyloaded (if not in view area, will not be loaded), etc…

* Enabling Emmet : setting -> Emmet include Language -> add, item:javascript & value:javascriptreact
* To apply css at global level, use it in \_app.js file, we cant use pure css type file in other js file.

Style Jsx

      <style jsx>

        {`

          .dummy{

            color:red;

          }

        `}

      </style>

Global level

      <style jsx global>

        {`

          .dummy{

            color:red;

          }

        `}

      </style>

NextJs uses 2 techniques for PreRendering

1.Server side rendering

2.Static site generation

**1.Server Side Rendering**

export async function getServerSideProps(context) {

    let data = await fetch('http://localhost:3000/api/blogs');

    let allBlogs= await data.json();

  return {

    props: {allBlogs}, // will be passed to the page component as props

  }

}

**2.Static site generation**

getStaticProps() method for loading all db datas or props

getStaticPaths() is for mentioning dependcy files or paths (as uderstood)

//from [slug].js

export async function getStaticPaths(context) {

  //how many dynamic routes are present

  return {

    paths: [

      {params: {slug: 'how-to-learn-javascript'}},

      {params: {slug: 'how-to-learn-nextjs'}},

    ],

    fallback: true

  }

}

export async function getStaticProps(context) {

  const {slug} = context.params;

  let myBlog = await fs.promises.readFile(`blogdata/${slug}.json`,'utf-8');

  return {

    props: {myBlog: JSON.parse(myBlog)}, // will be passed to the page component as props

  }}

Ex2

//from blog.js

export async function getStaticProps(context) {

  let dirData = await fs.promises.readdir("blogdata");

  let totalBlogsLength = dirData.length;

  let allBlogs = [];

  for (let index = 0; index < 5; index++) {

    const item = dirData[index];

    let myFile = await fs.promises.readFile(`blogdata/${item}`,'utf-8');

    allBlogs.push(JSON.parse(myFile))

  }

  return {

    props: {allBlogs, totalBlogsLength}, // will be passed to the page component as props

  }

}

2.1 Change configuration in next.config.js

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

  reactStrictMode: true,

  swcMinify: true,

  //so that static generated file can reload on reload button click (generated html file will be come inside a folders(about/about.html))

  trailingSlash:true

}

module.exports = nextConfig

2.2 Add a script to export the project in package.json

  "scripts": {

    "dev": "next dev",

    "build": "next build",

    "start": "next start",

    "lint": "next lint",

    "export": "next build && next export"

  },

Next step is >yarn export an “out” folder will be created which will be having static generated .html files.

Note: \*We cant use <Image> tag of nextjs if we want to generate static site. We can use normal <img> tag.

Rendering HTML text using dangerouslySetInnerHTML

  function createMarkup(htmlContent) {

    return { \_\_html: htmlContent};

}

function MyComponent() {

let htmlContent = “<html><div><h1>Hello world</h1></div></html>”

    return <div dangerouslySetInnerHTML={createMarkup(htmlContent)} />;

Using Fetch

let response = await fetch('http://localhost:3000/api/postcontact',{

      method: 'POST',

      headers: {

        'Content-Type': 'application/json'

      },

      body: JSON.stringify(reqBody),

    });

    let data = await response.text();

Git Removing a file from staging area

File will be removed from the index tracking our git project

> git rm -r --cached "folderName" (or) git –cached “folderName”

**Codes Wear Project**

TailWind Css installation in nextJs: <https://tailwindcss.com/docs/guides/nextjs>

Tailwind css extension: Tailwind CSS intelliSense

Note: [dynamicRoute].js

Ready made tailwind Blocks: <https://tailblocks.cc/>

Looping inside a Ojbect

Ex: let obj = { x:{ a:”a”},y:{b:”b”}}

Object.keys(obj).map((item)=>{..}); //here item will be having its object

Using Style inside div (React)

              style={{

                background: "linear-gradient(to right, #ee7724, #d8363a, #dd3675, #b44593)"

              }}

Connecting to MongoDB

>yarn add mongoose

# Create models or Schemas ex:

const mongoose = require('mongoose');

const OrderSchema = new mongoose.Schema({

    userId: {type:String, required: true},

    products:[ {

            productId: {type: String},

            quantity: {type: Number, default:1}

        } ],

    address: {type:String, required:true},

    amount: {type:Number, required:true},

    address: {type:String,default:"Pending", required:true},

},{timestamps: true});

export default mongoose.model("Order", OrderSchema);

# Create middleWare

import mongoose from "mongoose";

const connectDb = handler => async (req, res) =>{

    if(mongoose.connections[0].readyState){

        return handler(req,res);

    }

    await mongoose.connect(process.env.MONGO\_URI);

    return handler(req, res);

}

export default connectDb;

#Create the api ex1: pages>api>getProducts.js

import Product from "../../models/Product";

import connectDb from "../../middleware/mongoose";

const handler = async (req, res)=>{

    let products = await Product.find();

    res.status(200).json({products});

}

export default connectDb(handler);

ex2: pages>api>addProducts.js

import Product from "../../models/Product";

import connectDb from "../../middleware/mongoose";

const handler = async (req, res)=>{

    if(req.body.method == 'POST'){

        for(let i=0; i<req.body.length; i++){

            let product = new Product({title: req.body[i].title,

                slug: req.body[i].slug,

                desc: req.body[i].title,

                img: req.body[i].title,

                category: req.body[i].title,

                size: req.body[i].title,

                color: req.body[i].title,

                price: req.body[i].title,

                availableQty: req.body[i].title

                });

            await product.save();

        }

    }else{

        res.status(400).json({error:"This method now allowed"});

    }

}

export default connectDb(handler);

Error!!! - OverwriteModelError: Cannot overwrite `Product` model once compiled.

Reason: The first time you call the model creation function, mongoose stores the model under the key you provide (e.g. 'users'). If you call the model creation function with the same key more than once, mongoose won't let you overwrite the existing model.

Solution: checking if the model exists then use it, else create it.

const mongoose = require('mongoose');

const ProductSchema = new mongoose.Schema({

    title: {type:String, required: true},

    slug: {type:String, required:true, unique:true},

    desc: {type:String},

    img: {type:String},

    category: {type:String, },

    size: {type:String},

    color: {type:String},

    price: {type:Number, required:true},

    availableQty: {type:Number, required:true},

},{timestamps: true});

export default mongoose.models.Product || mongoose.model("Product", ProductSchema);