React Project – Beach Resort Takeaways

Install Packages   
1. React-Icons

2. React Router DOM

React-router-dom is ill-suited to Netlify. To fix that type, ‘react-router-dom an netlify’ and get the redirect file from ‘slightedgecoder.com’. We need to grab that file and it needs to be in the public folder.

Create images folder and data.js file in src folder. We can also import Router in index.js to free up the space.

Route parameters: ‘/rooms/:slug’ :slug can be anything

Rendering Error: Route component={Error}

Visit react-icons documentation. uses font-awesome

***Create Toggle state isOpen as false and method to toggle the navbar using setState***

***can use <ul> to render links however we need to use textDecoration as none in css***

NavBar: check css for handleToggle and show/hide. can do conditional rendering

can pass props as a className with curly braces

pass default props and proptypes which is a validation

We can achieve the same output with one condition i.e. by passing defaultProps in the hero component or by passing the css className to hero component in Home.js (Check hero and home component)

Pass props in components

1st part:

The first part was focused on importing images, data. Created components and routes.

Created navigation bars. Created a reusable component called Hero.js where we passed children and hero as a prop as well as defaultProps for css. We can also pass props to a className.

Resused Hero component to all pages component by importing to the component. What happens is when we pass one component to other component it takes all its attributes.

Also created a reusable banner component by passing three props(children, title, subtitle). To access children, pass children as prop, title, and subtitle. Now we can pass reuse banner for different components. Also created a link to direct users to different routes.

**Services Component to display services offered:**

Created a state with list of objects of arrays as well as react icons and used a map method to display items. Also used section and article html tags.

**loading component:**

imported a circular gif and rendered it in the component. Suited for data fetching.

**( : ) stands for alias**

**Context API**

Use curly braces in order to access the value which is passed as a prop in context provider.

In order to access all the data from the state of context.js we need to destructure value={{...this.state}} with double curly braces in our Provider. To access it in class based component, create static context type and access it using {value}

**Destructuring in Context API**

In Context API we need to import the data that we are going to use throughout the project. By importing we will receive a raw data/nested objects which needs to be accessed. In order to do that, first create a state with empty array. Now in order to access the data we need to format the data/map it over the data for our convenience which can be achieved by creating a method. If we don’t do this we won’t be able to access the data. Once we receive the data we need to update it to componentDidMount hook.

{Slug} we could do the id but we will have to have some kind of a number

React router gives us a nifty way to access the parameter or props i.e. match.

**React logics:**

if statement

conditional rendering

JS higher order functions (filter, map, sort, forEach)

Check slug, singleroom

**Styled Components called styledhero**

Passing props, created a separate styled component as header and passed props to background image. Exported to singleroom component as div.

**Accessing context API in functional components**

When using class, we have to access the context using static.

When using functional component, we have to access it through {value}

Context can also be accessed through a higher order component. Higher order components are functional components which are functions within a function.

**Filtering in React (filter.js component)**

Create a state in Context API for default values such as minPrice, maxPrice, maxSixe, minSize, etc. set maxPrice by obtaining it from the data in componentDIdMount using Math.max method from the rooms array to map and get the max price and pass it to the state because the max price already exists in the data.

Create handleChange event handler to grab the values from contextAPI

In RoomsFilter we are using useContext hook to access the data from the ContextAPI. For that we need to import the useContext from react and the roomContext from ContextAPI. After that we need to create a variable to access it such as const context = useContext(RoomContext). The same thing can be achieved through rendering component and higher order components.

After that we need to destructure what we need from the state.

To create a filter, use form html tags. controlled Input form for filter.

We can use select html and options to let users select from the lists manually but it is better that we do it from the by creating a function and returning only the unique values. (Filterrooms.js)

return [...new Set(items)]

Set only accepts unique values

**In order to get unique values from the data, we need to use new Set() Javascript syntax.**

**After that we need to create a function to getUnique values by mapping the items with value.**

**Now create a variable to look for type in rooms array. and add all types into the array and map it to jsx.**

const value = e.type === 'checkbox' ? target.checked: target.value

If the type is checkbox than target this or get the value of target checked. this.setState() has a callback function.

**Logic for filtering rooms (4:17:58):** The logic for filtering rooms is in the handlechange method where we target the value and name and based on the type we ll get the information and set the state accordingly.

To dynamically check the names throughout the application, in setState assign arrays.

Guests options:

When we have to deal with numbers, it is important that we check whether we get back string or a number. If we get a string, it is necessary to parse it to a number.

Checkout ROOMS FILTER .js very important

**ContentFul**