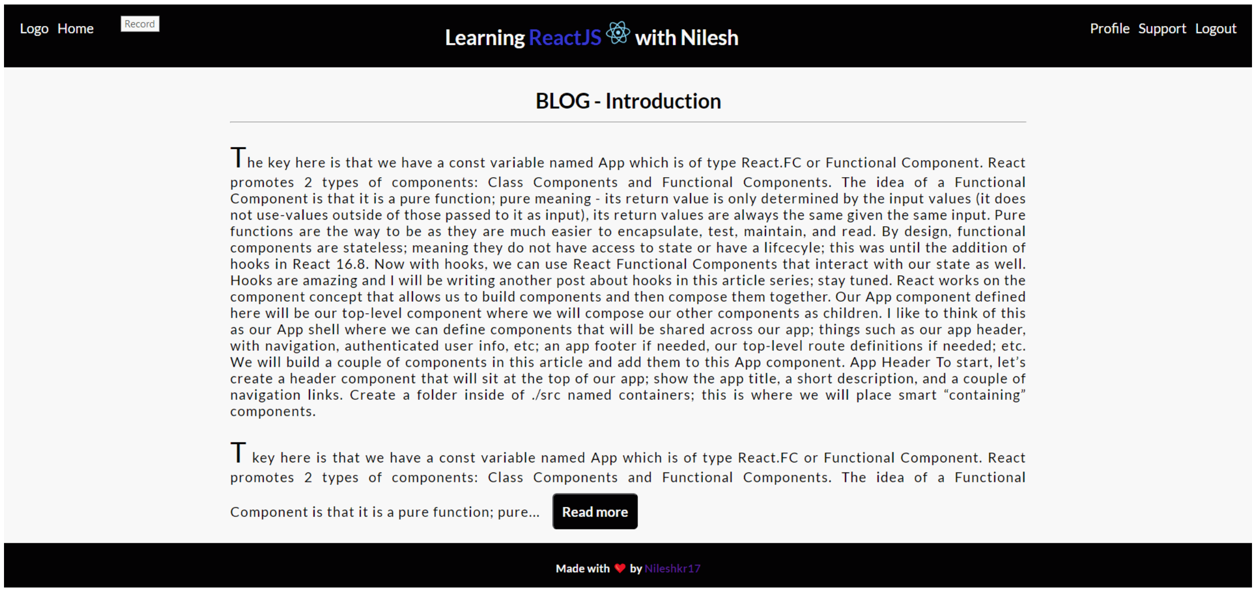
Assignment 2

Nilesh_20BECD76
May 30, 2023

Link: https://github.com/nileshkr17/PrepsLab/tree/main/Day%2002

Output:



BLOG - Introduction

The key here is that we have a const variable named App which is of type React.FC or Functional Component. React promotes 2 types of components: Class Components and Functional Components. The idea of a Functional Component is that it is a pure function; pure meaning - its return value is only determined by the input values (it does not use-values outside of those passed to it as input), its return values are always the same given the same input. Pure functions are the way to be as they are much easier to encapsulate, test, maintain, and read. By design, functional components are stateless; meaning they do not have access to state or have a lifcecyle; this was until the addition of hooks in React 16.8. Now with hooks, we can use React Functional Components that interact with our state as well. Hooks are amazing and I will be writing another post about hooks in this article series; stay tuned. React works on the component concept that allows us to build components and then compose them together. Our App component defined here will be our top-level component where we will compose our other components as children. I like to think of this as our App shell where we can define components that will be shared across our app; things such as our app header, with navigation, authenticated user info, etc; an app footer if needed, our top-level route definitions if needed; etc. We will build a couple of components in this article and add them to this App component. App Header To start, let's create a header component that will sit at the top of our app; show the app title, a short description, and a couple of navigation links. Create a folder inside of ./src named containers; this is where we will place smart "containing" components.

T key here is that we have a const variable named App which is of type React.FC or Functional Component. React promotes 2 types of components: Class Components and Functional Components. The idea of a Functional Component is that it is a pure function; pure...

HTML CODE:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Article</title>
    <link rel="stylesheet" href="style/style.css" />
</head>

<body>
    <div class="sticky">
        <div class="left">
            <li>Logo</li>
            <li>Home</li>
        </div>
        <div class="right">
            <li>Profile</li>
            <li>Support</li>
            <li>Logout</li>
        </div>
        <h2>Learning <div class="react-js">ReactJS</div>
            <img
src="https://img.icons8.com/color/48/000000/react-native.png"
class="fa-spin" width="30px" alt="react-icon" />
             with Nilesh</h2>
    </div>
    <div class="article">
        <div class="header">BLOG - Introduction</div>
        <hr />
        <div class="content">
            <p> The key here is that we have a const variable named
App which is of type React.FC or Functional
                Component. React promotes 2 types of components: Class
Components and Functional Components. The idea of
                a Functional Component is that it is a pure function;
pure meaning - its return value is only determined
                by the input values (it does not use-values outside of
those passed to it as input), its return values
                are always the same given the same input. Pure
functions are the way to be as they are much easier to
                encapsulate, test, maintain, and read. By design,
functional components are stateless; meaning they do
                not have access to state or have a lifcecyle; this was
until the addition of hooks in React 16.8. Now
                with hooks, we can use React Functional Components
that interact with our state as well. Hooks are
                amazing and I will be writing another post about hooks
in this article series; stay tuned. React works
                on the component concept that allows us to build
components and then compose them together. Our App
                component defined here will be our top-level component
where we will compose our other components as
                children. I like to think of this as our App shell
where we can define components that will be shared
                across our app; things such as our app header, with
navigation, authenticated user info, etc; an app
                footer if needed, our top-level route definitions if
needed; etc. We will build a couple of components
```

```
                in this article and add them to this App component.
App Header To start, let’s create a header component
                that will sit at the top of our app; show the app
title, a short description, and a couple of navigation
                links. Create a folder inside of ./src named
containers; this is where we will place smart “containing”
                components. <br>

            </p>
            <p> T key here is that we have a const variable named App
which is of type React.FC or Functional
                Component. React promotes 2 types of components: Class
Components and Functional Components. The idea of
                a Functional Component is that it is a pure function;
pure<span id="dots">...</span>
<span id="more"> </span>
<button onclick="myFunction()" id="myBtn">Read more</button>
            </p>
        </div>
    </div>

    </div>

    <div class="footer">
        <h5>Made with ❤️ by <a
href="https://github.com/nileshkr17">Nileshkr17</a>  </h5>
    </div>
</body>
<script src="script/script.js"></script>
</html>
```

Javascript:

```
function myFunction() {
    const dots = document.getElementById("dots");
    const moreText = document.getElementById("more");
    const btnText = document.getElementById("myBtn");

    if (dots.style.display === "none") {
      dots.style.display = "inline";
      btnText.innerHTML = "Read more";
      moreText.style.display = "none";
    } else {
      dots.style.display = "none";
      btnText.innerHTML = "Read less";
      moreText.style.display = "inline";
    }
  }

  document.getElementById("clickable").addEventListener("click",
function(){
    document.getElementById("more").innerHTML = " meaning - its return
value is only determined by the input values (it does not use-values
outside of those passed to it as input), its return values are always
the same given the same input. Pure functions are the way to be as
they are much easier to encapsulate, test, maintain, and read. By
design, functional components are stateless; meaning they donot have
access to state or have a lifcecyle; this was until the addition of
hooks in React 16.8. Nowwith hooks, we can use React Functional
Components that interact with our state as well. Hooks areamazing and
I will be writing another post about hooks in this article series;
stay tuned. React workson the component concept that allows us to
build components and then compose them together. Our Appcomponent
defined here will be our top-level component where we will compose our
other components aschildren. I like to think of this as our App shell
where we can define components that will be sharedacross our app;
things such as our app header, with navigation, authenticated user
info, etc; an appfooter if needed, our top-level route definitions if
needed; etc. We will build a couple of componentsin this article and
add them to this App component. App Header To start, let’s create a
header component that will sit at the top of our app; show the app
title, a short description, and a couple of navigationlinks. Create a
folder inside of ./src named containers; this is where we will place
smart “containing”components. ";
});
```

CSS

```
@import
url('https://fonts.googleapis.com/css2?family=Lato:wght@100;300;400;70
0;900&family=Open+Sans:ital,wght@0,400;1,800&family=Orbitron:wght@400;
500;600;700;800;900&family=Roboto:wght@500&family=Rubik+Iso&family=Ubu
ntu&display=swap');

*{
    padding: 0;
    margin: 0;
    font-family: Lato, sans-serif;
    box-sizing: border-box;
    list-style: none;
    text-decoration: none;

}
.react-js{
    color: rgb(56, 56, 206);
    display: inline;
}
.content>p{
    color: #000;
}
.profile{
    position: relative;
}
.tet{
    position: absolute;
    top: 200px;
    margin: 0 30%;
    color: aliceblue;
    font-weight: 600;
    font-size: 4rem;
}

.sticky{

    position: fixed;
    top: 0;
    align-items: center;
    width: 100%;
    z-index: 1;
    background-color: black;
    text-align: center;
    padding: 20px;
    color: white;
}
h3{
    font-size: 2rem;
    font-weight: 600;
    color: aliceblue;
    margin: 0 30%;
    padding: 10px;
}
```

```
.content{
    position: relative;
    top: 0;
    width: 100%;

}
.footer{
    position: fixed;
    bottom: 0;
    width: 100%;
    background-color: black;
    color: white;
    padding: 20px;
    text-align: center;
}

button{
    background-color: black;
    color: white;
    padding: 10px;
    border-radius: 5px;
    cursor: pointer;
    font-weight: 600;
    font-size: 1rem;
    margin: 10px;
}
/* .submission{
    margin: 0 auto;
    width: 50%;
    height: 75vh;
    padding: 10px;
    background-color: whitesmoke;
    margin-top: 5rem;
} */
.article{
    margin: 0 auto;
    width: 65%;
    padding: 10px;
    background-color: whitesmoke;
    margin-top: 5rem;

}
.article>.header{
    text-align: center;
    font-size: 1.5rem;
    font-weight: 600;
    padding: 10px;
}

p{
    margin-top: 20px;
    color: black;

    text-align: justify;
    letter-spacing: 1px;

}
p::first-letter{
    font-size: 200%;
    color: #000;
}

h5{
    color: white;
    position:relative;
}
.left{
    float: left;
    display: flex;
    justify-content: space-around;
}
.right{
    float: right;
    display: flex;
    justify-content: space-around;
}

.left>li{
    margin-right: 10px;
}
.right>li{
    margin-left: 10px;
}
#more {
    display: none;
}
```