Setup your project

Perform the necessary installation

1. Install nodejs

- Go to the website https://nodejs.org/en/
- Go to Downloads and use the installer for your respective operating system

2. Install a code editor of your choice

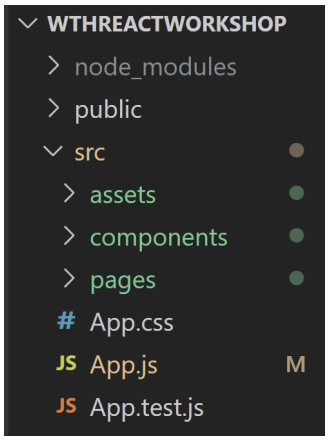
- install visual studio code or any editor of your choice: https://code.visualstudio.com/download

Once go to your folder directory in your command prompt/terminal and call the following commands

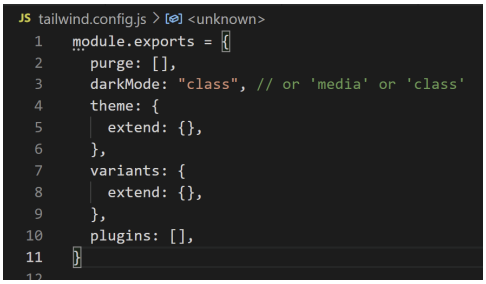
1. npx create-react-app wthreactworkshop
2. cd wthreactworkshop
3. Go to tailwind documentation and follow the instructions here: https://tailwindcss.com/docs/guides/create-react-app
4. npm install react-icons –save: https://react-icons.github.io/react-icons/

Create three folders in your src directory

1. components
2. images
3. pages



Go to tailwind.config.js and change darkMode to be “class” (this will allow us to configure dark mode for our webpage later)



Javascript basics

Some important syntax and functions to know:

1. variables

```
let x = 0;
```

2. arrow functions

```
const functionName = (arguments) => {
  return
}
```

3. ternary operator

- the ternary operator, or question mark in this case checks against a condition, if the condition is true, it will call the value or function to the left of the semicolon, else if the condition is false, it will call the value or function to the right of the semicolon

```
condition ? "true condition": "false condition"
```

4. template literals

- template literals are enclosed by two backticks `` and the variable is wrapped by ${}

```
`${variableName}`
```

5. Arrays

- A javascript array allows you to store multiple values enclosed in square brackets separated by commas

```
let thisIsAnArray = ["apples", "oranges", "lemons"]
```

6. Objects

- A javascript object stores key value pairs and is defined using a set of curly brackets {}

```
let thisIsAnObject = {
  fruit: "apple",
  colour: "red",
  cost: 1.0
}
```

7. links you can read to learn more about the fundamentals

- variables: https://www.w3schools.com/js/js_variables.asp
- arrow functions: https://www.w3schools.com/-Js/js_arrow_function.asp
- ternary operator: https://www.programiz.com/javascript/ternary-operator
- template literals: https://www.w3schools.com/-JS//js_string_templates.asp
- arrays: https://www.w3schools.com/js/js_arrays.asp
- objects: https://www.w3schools.com/js/js_objects.asp

Web Dev Basics

How to define a html element:

1. an element is defined by using opening tags <> and closing tags </>

- for example, if we want to define a div html element (the most basic building block):

```
<div>
   This is my first html element yay :D
</div>
```

Three types of elements:

1. block
- by default takes up the full width of parent

2. inline
- only take up as much space as required
- setting height and width have no effect on this element

3. inline-block
- combines both properties of inline and block
- only takes up as much space as element
- can set height and width on element

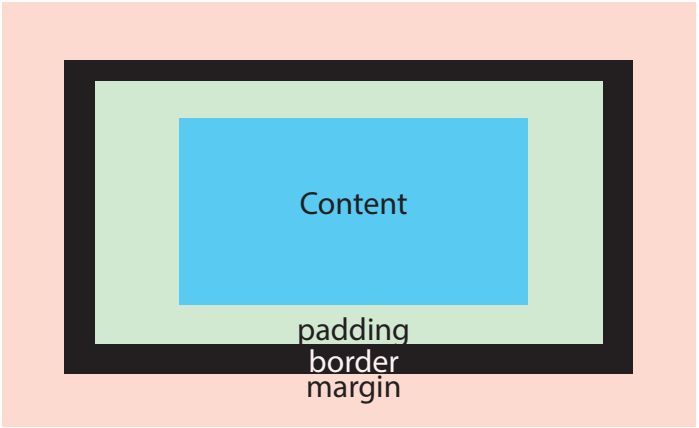
To learn more about block, inline and inline-block:
https://www.youtube.com/watch?v=x_i2gga-sYg

Box Model (padding, margin, border, content):

1. Every html element takes up space, and we can configure the space and positioning of the element using the box model

2. Box Model comprises of 4 parts:

- content: space which your elements take, and htmls elements, text and images come here
- padding: space around the content
- border: wraps both margin and padding, can serve as an outline for your block
- margin: space around the block (outside of border), can be used to provide spacing from other elements in your webpage



CSS Basics

classes

1. to give your html elements styling, you can give your html elements a class name and add the corresponding style in a separate stylesheet and link to the class name using a dot ".":

```
<div class="text-blue" >
   I have a class name of blue
</div>

// in separate stylesheet
.text-blue{
   color: blue;
}
```

in React, we use className instead of class

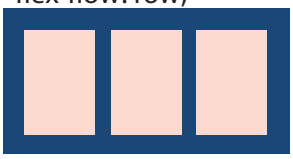
flexbox

1. flexbox (flexible box) provides a very useful toolkit for you to format your web pages

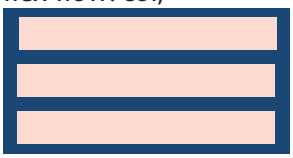
- very useful page for understanding flexbox:
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

2. set parent element to display flex and set the flex-flow to row or col to arrange child elements along the elements' row or column

display: flex;
flex-flow: row;

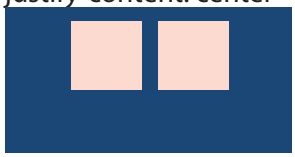


display: flex;
flex-flow: col;



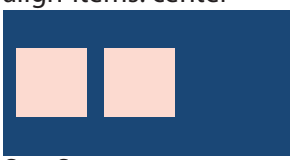
3. centering elements using justify-items: center and align items: center

flex-flow: row
justify-content: center

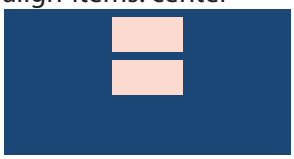


flex-flow: column
justify-content: center

flex-flow: row
align-items: center

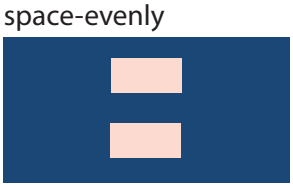


flex-flow: column
align-items: center



flex-flow: row
justify-content:
space-evenly

flex-flow: column
justify-items:
space-evenly



Tailwind CSS

Tailwind CSS makes use of classes to implement your styles

https://tailwindcss.com/docs

Flexbox:

- flex: make element a flexbox
- flex-row: flexbox in horizontal direction
- flex-col: flexbox in vertical direction
- flex-1: grow and shrink, but ignore initial size
- flex-auto: grow and shrink automatically in the flexbox, but account for initial size
- justify-center: same as justify-content: center
- items-center : same as align-items: center
- self-end: bring a child element to the end of the flexbox

https://v1.tailwindcss.com/docs/flex-direction
https://v1.tailwindcss.com/docs/flex

Text:

- text-center: center text
- text-white: set text to color (this case white), can change white to any available colour in tailwind

https://v1.tailwindcss.com/docs/text-color#class-reference

- text-bold
- text-xs: change text size (change xs to any available size)

https://v1.tailwindcss.com/docs/font-size#class-reference

Spacing:

https://tailwindcss.com/docs/padding

- p-5: add a padding of 5 (change 5 to any number supported by tailwind)
- px-5: add a padding of 5 to both left and right padding
- py-5: add a padding of 5 to top and bottom padding
- pt-5/pb-5/pr-5/pl-5: add padding of 5 to top, bottom, right and left respectively

https://tailwindcss.com/docs/margin

- m-5: add margin of 5
- mx-5: add margin of 5 to both left and right margin
- my-5: add margin of 5 to both top and bottom margin
- mt-5/mb-5/mr-5/ml-5: add margin of 5 to top, bottom, right and left respectively

Tailwind CSS(continued)

Background color

https://tailwindcss.com/docs/background-color

bg-black: set background colour to black
bg-white: set background colour to white
bg-transparent: set background colour to transparent
bg-blue-100: can change number to adjust shade of blue (in increments of 100)
etc

Hover

https://tailwindcss.com/docs/hover-focus-and-other-states

1. to add a hover effect, do hover:<insert class name here>, for example, if we want a button to be green when not hovered and blue when hovered, it will look like the following:

```
<button className="bg-green-400
hover:bg-blue-400"> button </button>
```

Animation

https://tailwindcss.com/docs/animation

1. tailwind provides some animation we can use out of the box, examples of possible animations include:
animate-spin
animate-bounce
...and more

Dark mode

https://tailwindcss.com/docs/dark-mode

1. dark mode can be easily set in tailwind by adding a dark: <insert-class-style-here> and when we set the class of the div enclosing our application, it will change all elements to their "dark:" styling. for example:

```
- this will be blue
<div className="App">
  <div className="bg-blue-400 dark:bg-black>
    I will be blue
  </div>
</div>

- this will be black
<div className="App dark">
  <div className="bg-blue-400 dark:bg-black>
    I will be black
  </div>
</div>
```

this will only work when darkMode is set to class in tailwind.config.js (refer to page 1 of this notes)

React Setup

React allows us to create components, pass down the state of our application in a seamless way

https://reactjs.org/docs/getting-started.html

Setting up your React application:

This is alread described in page 1 but it will be reiterated here

- npx create-react-app nameOfProject

Files that you will see in your create-react-app application

-- node modules: contains all dependency code (including those that you install), most of the times dont need to touch

-- public folder: contains the index.html file which is used as the template file for your app

-- src folder: all code relating to your application go here

---- App.js: main home page code, you will add all your React components to this main page

---- App.css: css page for the main app

-- package.json: stores metadata of the project as well as the list of dependency packages

-- package-lock.json: stores an exact versioned dependency tree, will regenerate if package.json change. package.json only contains direct dependencies and not nested dependencies

Folders that we will add in our React project:

1. assets: we will store our images here
2. components: write components here
3. pages: write pages here

Why do we need these folders?

In a web application we will often have multiple pages which users toggle around, and within a page we will often have components. Within components, we may want to include assets such as images or svgs into our application

React basics

Previously, we mentioned that our web application is built up of pages and components. In order to build these pages and components, we need to add in the basic block of React (or a React component):

- A basic react component can be defined using a javascript arrow function (described in the Javascript basics section). This is demonstrated below:

src > components > JS Component.js > ...

```
1   const Component = props => {
2       return(
3           <div>
4               Content of component
5           </div>
6       )
7   }
8
9   export default Component
```

A few things to note when defining this component:

1. Component takes in an argument props, we will learn what props is later
2. after we create a component we must export the component using an export default so that other components can use it
3. Component names should be capital letters

Best practices in defining components:

1. name the component the same as the file
2. one component in one file
3. Create a component if you feel like something is repeated a few times

Passing in props to our component

To pass in props is to pass information to our component. Oftentimes, we have components with values that can change, hence we handle this flexibility with props

To handle props, we take these steps:

1. define the component using props
2. pass in the information in the parent component

See next page for example on props

React basics

...continued
assuming we have a Button component, and we want to keep the text inside flexible (so that we can reuse for many different buttons) , we use the props with the text properties to account for different values

```
const Button = props => {
    return(
        <button>
            {props.text}
        </button>
    )
}

export default Button
```

We now call this component in our main page, passing in the props text when defining the component.

There are two steps to using a component:

1. import the component using the format:
import ComponentName from "./components/ComponentName"
2. use the component name by enclosing it in </>:
<ComponentName/>

```
import Button from "./components/Button"

function App() {
  return (
    <div>
      <Button text="Click me"/>
    </div>
  );
}

export default App;
```

Creating a component multiple times using map

Imagine we have a card component which we have created which we intend to create 4 times with two prop values

```
const Card = props => {
  return(
    <div>
      <div>{props.icon}</div>
      <div>{props.text}</div>
    </div>
  )
}
export default Card
```

See next page for continued example on map

React basics

...continued
though it may be tempting to copy and paste the component multiple time such that it looks something like this:

```
import Card from "./components/Card"
import { FaCode, FaSchool, FaMoon } from "react-icons/fa"
import { IoSunny } from "react-icons/io5"

const Homepage = props => {
    return (
      <div>
        <Card icon={<FaCode/>} text="card 1"/>
        <Card icon={<FaSchool/>} text="card 2"/>
        <Card icon={<FaMoon/>} text="card 3"/>
        <Card icon={<IoSunny/>} text="card 4"/>
      </div>
    );
  }

export default Homepage;
```

Using this approach will slowly clutter the application and we will find it hard to manage the application as it scales, as such a better way of reusing multiple components is through storing the information in an array and using a map to create multiple values

two steps is used to achieve this:

1. create an array storing your information in javascript objects
2. use the map function and map the information into the props of your component

```
import Card from "./components/Card"
import { FaCode, FaSchool, FaMoon } from "react-icons/fa"
import { IoSunny } from "react-icons/io5"

const Homepage = props => {
    let cardsInfo = [
      {
        icon: <FaCode/>,
        text: "card 1"
      },
      {
        icon: <FaSchool/>,
        text: "card 2"
      },
      {
        icon: <FaMoon/>,
        text: "card 3"
      },
      {
        icon: <IoSunny/>,
        text: "card 4"
      }
    ]
    return (
      <div>
        {cardsInfo.map(card => <Card icon={card.icon} text={card.text}/>)}
      </div>
    );
  }

export default Homepage;
```

more information on the map function can be found here:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

React basics

State using useState

A state in a web application could refer to any sort of state switching that our applciation takes, for example, one kind of state our application could have is whether our application is in light mode or dark mode.

as such, in the following example, we can set a state of isDarkMode using a true or false value

```
import { useState } from "react"
import HomePage from "./pages/HomePage"


function App() {
  let [isDarkMode, setIsDarkMode] = useState(false)
```

in this example, the default value or starting state of our application is not dark mode, and hence we set the default value in the useState command to be false.

we also have a variable setIsDarkMode, this command is used whenever we want to change the value of our state, isDarkMode.

for example, we may want to create a button that allows us to toggle between light mode and dark mode, and hence we create a function toggleDarkMode shown below

```
import { useState } from "react"
import HomePage from "./pages/HomePage"


function App() {
  let [isDarkMode, setIsDarkMode] = useState(false)

  let toggleDarkMode = () => {
    setIsDarkMode(!isDarkMode)
  }
```

the exclamation mark that we see !isDarkMode is a way of saying "not", so what this function is doing is that when we press the button and isDarkMode is equals to false, not false will equals to true and hence we set the value of isDarkMode to be true. Conversely, if isDarkMode = true, then !isDarkMode = false and hence we will setIsDarkMode to be false.

React basics

Bringing it all together with useState

to toggle dark mode we need to change the class of our app to be "dark" when isDarkMode = true, and "" when isDarkMode = false, hence we can use ternary operators and template literals explained in javascript basics to help us

```
import { useState } from "react"
import HomePage from "./pages/HomePage"


function App() {
  let [isDarkMode, setIsDarkMode] = useState(false)

  let toggleDarkMode = () => {
    setIsDarkMode(!isDarkMode)
  }

  return (
    <div className={`${isDarkMode ? "dark": ""}`}>
      <HomePage isDarkMode={isDarkMode} toggleDarkMode={toggleDarkMode}/>
    </div>
  );
}

export default App;
```

lastly we use pass in both isDarkMode and toggleDarkMode function as props to homepage which will connect to a button that will help us to toggle between dark mode and light mode. we will also toggle between a sun and moon icon based on the state of dark mode in our app within the button!

In src/pages/HomePage.js

```
import { FaMoon } from "react-icons/fa"
import { IoSunny } from "react-icons/io5"


const HomePage = props => {
  return (
    <div>
      <button onClick={props.toggleDarkMode}>
        {props.isDarkMode ? <IoSunny/>:<FaMoon/>}
      </button>
    </div>
  )
}
export default HomePage
```

The concepts we covered in this tutorial include:
1. how to create as react component
2. how to pass down props to components
3. how to reuse multiple compoenents using map
4. how to use useState to toggle the state of our application
5. how to write a function to change the state of our useState variable
6. using the onClick function of the button to toggle our state
7. How to setup and use tailwind to setup your css using className

All the Best For the Hackathon, may the best hack win! :D

Feel free to contact us using telegram @clarence34 and @brycerice