**This past week I was able to dive deeper into React and React native fundamentals and really explored how things like components, state, props, rendering, event handling are done in react. It was different from just raw HTML/CSS and it was interesting to see everything coming into play. The way I practiced my skills was by making a basic to list page, following youtube tutorials, react documentations and online guides. I will break down everything I was able to practice and cover and document the code that I was able to practice writing to really understand what was going on.**

**Components:**

```
import React from 'react'
export default function Todo({ todo, toggleTodo }) {
 function handleTodoClick() {
  toggleTodo(todo.id)
 }

 return (
  <div>
   <label>
    <input type="checkbox" checked={todo.complete} onChange={handleTodoClick} />
    {todo.name}
   </label>
  </div>
 )
}

import React from 'react'
import Todo from './Todo'
export default function TodoList({ todos, toggleTodo }) {
 return (
  todos.map(todo => {
   return <Todo key={todo.id} toggleTodo={toggleTodo} todo={todo} />
  })
 )
}
```

**I created these two components todo and todo list. Todo represents an item that is to be added to the list. It has the ability to be checked off and has a function to click that will toggle its state.**

**I also created the todolist function where I passed in todos and toggleTodo. It was interesting to learn about the map function that would allow me to build collections of items, and allow me to create a key, a toggle on/off, and add a todo object.**

**Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail.**

**These components were interesting to construct in react and it was intriguing to see the differences between vanilla javascript and react.**

**States, Effects**

```
import React, { useState, useRef, useEffect } from 'react';
import TodoList from './TodoList'
import uuidv4 from 'uuid/v4'

const LOCAL_STORAGE_KEY = 'todoApp.todos'

function App() {
 const [todos, setTodos] = useState([])
 const todoNameRef = useRef()

 useEffect(() => {
  const storedTodos = JSON.parse(localStorage.getItem(LOCAL_STORAGE_KEY))
  if (storedTodos) setTodos(storedTodos)
 }, [])

 useEffect(() => {
  localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(todos))
 }, [todos])
```

**This was on my main app.js page that represented my root page. Here I experimented with States and Effects. I learned specifically that a state, when called, allows us to keep our local state in a function component. Specifically, in my instance, I was able to call the useState function at the start of the program by passing in an empty array because I have no todos. And then onwards, I would destruct the useState method into the arrays, one containing my todos, and another method that would allow me to change my todos. As a whole, the useState function is extremely powerful in letting me change and update my data, while maintaining a sense of order. Just to recap, It initially has a default value of just an empty array, and as user clicks and adds todos, it will re-render the page with my new todos and allow my webpage or app to meet the users demands of adding or clearing todos from the todolist page.**

**I also learned about the useEffect function, which more or less allowed me to preserve my data. Every time some sort of dependency is called the useEffect function would run and store my todos into local storage. Basically, when I refresh the page, my todos would not disappear from my todo list and would still remain on my page allowing a better user interface that will reduce frustration due to misclicks.**

**Main App/Root Function:**

```
function App() {
 const [todos, setTodos] = useState([])


 function toggleTodo(id) {
  const new Todos = [...todos]
  const todo = newTodos.find(todo => todo.id === id)
  todo.complete = !todo.complete
  setTodos(newTodos)
 }

 function handleAddTodo(e) {
  const name = todoNameRef.current.value
  if (name === '') return
  setTodos(prevTodos => {
    return [...prevTodos, { id: uuidv4(), name: name, complete: false}]
  })
  todoNameRef.current.value = null
 }

 function handleClearTodos() {
  const newTodos = todos.filter(todo => !todo.complete)
  setTodos(newTodos)
 }

 return (
  <>
    <TodoList todos={todos} toggleTodo={toggleTodo} />
    <input ref={todoNameRef} type="text" />
    <button onClick={handleAddTodo}>Add Todo</button>
    <button onClick={handleClearTodos}>Clear Complete</button>
    <div>{todos.filter(todo => !todo.complete).length} left to do</div>
  </>
 ) export Default App;
```

**All these things would reside in my function app, which would handle all user inputs and the text that they are expected to see. It involves several responding functions that will ensure that user input is met with the adequate and appropriate response to ensure that the user experience is good,  but also to ensure that the certain tasks that user needs to be updated are appropriately managed.**

**Despite its small size, I was able to learn a lot of core fundamentals of react and really understand its power in its scalability and reusability. With the idea of having many**

**components and also being able to have clear cut typed syntax, I am able to understand and debug the code a lot faster then just normal javascript,**

**Resources Used:**
**https://www.youtube.com/watch?v=0-S5a0eXPoc**
**https://www.youtube.com/watch?v=6I_4j_Uzc**
**https://reactjs.org/tutorial/tutorial.html**
**https://www.youtube.com/@WebDevSimplified**
**https://www.tutorialspoint.com/react_native/index.htm**
**https://reactnative.dev/docs/tutorial**