2023년-2기 정규 과정 CNU SW 아카데미
주간 학습 보고서

트랙명	☐ 백엔드	☑ 프론트엔드
학습 기간	23. 7. 3. (월) ~ 23. 7. 7. (금)	
학습 내용		
7월 3일 (월)	학습 시간	시작 09시 01분 ~ 종료 18시 03분 (총 9시간 2분)
	학습 구분	☐ 이론교육 ☑ 실습교육 ☐ 프로젝트 ☐ 특강/세미나/현장견학 특별교육 등
	학습 장소	☑ 정보화본부 강의실 ☐ 기타 (온라인)
	교수/강사/TA 명	프로그래머스 강의 8주차 React Basic Hooks
	학습내용(요약)	React Basic Hooks에 관련하여 학습하였다. - useState: 값이 변경되면 컴포넌트가 렌더링 된다. - useEffect: 컴포넌트가 Mount, Update, UnMount될 때 실행되는 함수를 의미한다. - useRef: current객체를 통해 DOM을 접근할 수 있고 지역변수로도 사용가능하다. 즉 값이 변경되도 컴포넌트가 렌더링되지 않는다. React의 리렌더링 조건을 확인하여 성능 최적화 관련한 학습을하였다. - useMemo: 계산의 return 값을 저장하는 방법이고 결과값의 종속성이 바뀌지 않으면 계속해서 재사용 가능하다. - React.memo: 컴포넌트 자체를 저장하는 방법이고 부모 컴포넌트가 전달한 props값의 상태변화에 따라 리렌더링이 결정된다. - useCallback: return 값이 아닌 함수 자체를 저장하여 컴포넌트 렌더링 되면 컴포넌트 내에 있던 함수가 새로 생성되면서 리렌더링 되므로 이를 막을 수 있다.
7월 4일 (화)	학습 시간	시작 08시 57분 ~ 종료 18시 00분 (총 9시간 3분)
	학습 구분	☐ 이론교육 ☑ 실습교육 ☐ 프로젝트 ☐ 특강/세미나/현장견학 특별교육 등
	학습 장소	☑ 정보화본부 강의실 ☐ 기타 (온라인)
	교수/강사/TA 명	프로그래머스 강의 8주차 React Component 구현
	학습내용(요약)	React에서 자주 사용하는 Component를 storybook과 함께 사용하여 구현하는 방법을 학습하였다. (1) Text Component - 폰트 스타일과 관련한 파라미터를 받고 fontStyle 객체에서 스타일을 설정한다. - block, paragraph에 따라 이중 삼항연산자를 통해 태그글 구분하도록 작성한다. - mark, code, delete를 각각 if문으로 두어 children이 중첩될 수 있도록 작성한다. - size의 타입이 number이면 inline style로 설정하고 string이면 Text.css에 정의해준 className으로 설정하도록 한다. - propTypes에서 onOfTypes([])은 둘 중 하나의 타입을 가질 수 있다는 것을 의미한다. (2) Header Component - level에 따른 heading tag를 정의한다. - level이 1보다 작고 6보다 큰 값을 전달받으면 예외처리를 h1으로 예외처리한다. - propTypes에서의 isRequired는 필수적으로 받아야하는 prop을 의미한다.

7월 5일 (수)	**학습 시간**	시작 09시 01분 ~ 종료 21시 23분 (총 12시간 22분)
	학습 구분	□ 이론교육 ☑ 실습교육 □ 프로젝트 □ 특강/세미나/현장견학 특별교육 등
	학습 장소	☑ 정보화본부 강의실 □ 기타 (온라인)
	교수/강사/TA 명	프로그래머스 강의 8주차 Search 기능 구현
	학습내용(요약)	React에서 자주 사용하는 기능인 검색 기능을 구현하는 법을 학습하였다. indexOf() - 문자열에서 특정 문자의 위치를 구할때 사용한다. - 특히 사용자가 검색하고 싶은 내용을 검색할 때 해당 키워드가 데이터와 일치하는지를 판단할 때 많이 사용한다. - 파라미터로 받은 문자열 혹은 문자가 연속적으로 일치하지 않으면 -1을 return한다. - 파라미터로 받은 문자열이 연속족으로 일치하면 0을 return한다. - 파라미터로 받은 문자가 존재하면 해당 index를 return한다.
7월 6일 (목)	**학습 시간**	시작 09시 20분 ~ 종료 22시 10분 (총 12시간 50분)
	학습 구분	□ 이론교육 ☑ 실습교육 □ 프로젝트 □ 특강/세미나/현장견학 특별교육 등
	학습 장소	☑ 정보화본부 강의실 □ 기타 (온라인)
	교수/강사/TA 명	프로그래머스 강의 8주차 React Custom Hooks
	학습내용(요약)	React에서 자주 사용하는 localStorage에 데이터를 저장하는 Custom Hook을 작성하는 법을 학습하였다.
7월 7일 (금)	**학습 시간**	시작 09시 20분 ~ 종료 19시 20분 (총 10시간 00분)
	학습 구분	□ 이론교육 ☑ 실습교육 □ 프로젝트 □ 특강/세미나/현장견학 특별교육 등
	학습 장소	☑ 정보화본부 강의실 □ 기타 (온라인)
	교수/강사/TA 명	프로그래머스 강의 8주차 Context API
	학습내용(요약)	React에서 전역적인 데이터를 관리할 때 사용하는 ContextAPI에 대해 학습하였다. - Redux와는 다르게 상태관리를 해주는 것이 아닌 상태를 전역적으로 공유해주는 기능만을 수행한다. - Context의 Provider와 Consumer를 사용해야한다. - 남용시 Consumer를 사용하는 컴포넌트들이 모두 렌더링 되므로 성능이 떨어질 수 있다.

붙임 주간 학습 상세 보고서 1부. 끝.

위와 같이 주간 학습을 성실히 수행하고 보고서를 제출합니다.

2023. 7. 7.

학생: 강병헌 (인)

CNU SW 아카데미 귀하

주간 학습 상세 보고서

학습주제	React Basic Hooks
학습 및 실습 내용	

React Basic Hooks

useState

- 값이 변경되면 컴포넌트가 렌더링 된다.

useEffect

- 컴포넌트가 Mount, Update, UnMount될 때 실행되는 함수를 의미한다.

```
// Mount
useEffect(() => {}, []) // 컴포넌트가 처음 렌더링 될 때 함수가 실행된다.

// Update
useEffect(() => {
    const elem = ref.current

    if (elem) { // depth update
      elem.addEventListener('mouseover', handleMouseOver)
      elem.addEventListener('mouseout', handleMouseOut)

      return () => {  // Unmount
        elem.removeEventListener('mouseover')
        elem.removeEventListener('mouseout')
      }
    }
  }, [ref, handleMouseOut, handleMouseOver])
// depth 값이 바뀌면 해당 함수가 실행된다.

// Unmount
useEffect(() => {
  ...
  return () => {
    ...
  }
}, []) // 컴포넌트가 사라질 때 return된 함수가 실행된다.
```

useRef

- current객체르 통해 DOM을 접근할 수 있다.
- 지역 변수로 사용 가능하다. ⇒ 값이 변경되면 컴포넌트가 렌더링 되지 않는다.

하위 컴포넌트에게 ref prop 전달

- React.forwardRef()함수를 사용하여 부모 컴포넌트의 ref를 prop으로 받을 수 있다.

```
// App.js
export default const App = () => {
  const inputRef = useRef()

  return (
    <>
      <Input ref={inputRef} />
    </>
  )
}

// Input.js
export default const Input = React.forwardRef((_, ref) => {
  return (
    <input ref={ref} />
  )
})
```

성능 최적화

리렌더링 조건

1. 현재 컴포넌트에서 상태가 변경될 때
2. 부모 컴포넌트로 받은 prop의 상태가 변경될 때
3. 부모 컴포넌트의 상태가 변경될 때

useMemo

- 계산의 return값을 저장하는 방법이다.
- 결과값의 종속성이 바뀌지 않으면 계속해서 재사용 가능하다.

```
// 의존성 배열에 있는 값이 업데이트 될 경우 해당 콜백함수가 실행된다.
const result = useMemo(() => sum(n), [n])
```

❗ React.memo

- 컴포넌트 자체를 저장하는 방법이다.
- 부모 컴포넌트가 전달한 props값의 상태 변화에 따라 리렌더링이 결정된다.
- 컴포넌트서 리렌더링이 필요한 상황에서만 리렌더링을 하도록 설정

```
const ChildComponent = React.memo(() => {
  return (
    <div>
      <span>Hello World!</span>
    </div>
  ))
}
```

❗ useCallback

- return 값이 아닌 함수 자체를 저장하는 방법이다.
- 컴포넌트가 렌더링 되면 컴포넌트 내에 있던 함수가 새로 생성되면서 리렌더링 되므로 이를 막아줄 필요가 있다.

```
const onChange = useCallback(() => {
  // function..
}, [])
```

학습주제	React Component 구현
학습 및 실습 내용	# Text ## Text.js ``` import './Text.css' import PropTypes from 'prop-types' const Text = ({ children, block, paragraph, size, strong, underline, delete: del, color, mark, code, ...props }) => { const Tag = block ? 'div' : paragraph ? 'p' : 'span' const fontStyle = { fontSize: typeof size === 'number' ? size : undefined, fontWeight: strong ? 'bold' : undefined, textDecoration: underline ? 'underline' : undefined, color, } if (del) { children = ~~{children}~~ } if (mark) { children = <mark>{children}</mark> } if (code) { children = <code>{children}</code> } return (<Tag className={typeof size === 'string' ? `Text--size-${size}` : undefined} style={{ ...props.style, ...fontStyle }} > {children} </Tag>) } ```

```
Text.propTypes = {
  children: PropTypes.node.isRequired,
  block: PropTypes.bool,
  paragraph: PropTypes.bool,
  size: PropTypes.onOfType([PropTypes.number, PropTypes.string]),
  strong: PropTypes.bool,
  underline: PropTypes.bool,
  delete: PropTypes.bool,
  color: PropTypes.string,
  mark: PropTypes.bool,
  code: PropTypes.bool,
}

export default Text
```

- 폰트 스타일과 관련한 파라미터로 받고 fontStyle 객체에서 스타일을 설정한다.
- block, paragraph에 따라 이중 삼항연산자를 통해 태그를 구분하도록 작성한다.
- mark, code, delete를 각각 if문으로 두어 children이 중첩될 수 있도록 작성한다.
- size의 타입이 number이면 inline style로 설정하고 string이면 Text.css에 정의해둔 className으로 설정되도록 작성한다.
- propTyeps에서 onOfTypes([])은 둘 중에 하나의 타입을 가질 수 있다는 것을 의미한다.

Text.css

```
.Text--size-small {
  font-size: 12px;
}

.Text--size-normal {
  font-size: 14px;
}

.Text--size-large {
  font-size: 18px;
}
```

- size의 타입이 string일 경우 해당 className으로 지정될 수 있도록 정의해놓는다.

Text.stories.js

```
import React from 'react'
import Text from '../components/Text'

export default {
  title: 'Component/Text',
  component: Text,
  argTypes: {
    size: { control: 'number' },
    strong: { control: 'boolean' },
    underline: { control: 'boolean' },
    delete: { control: 'boolean' },
    color: { control: 'color' },
    block: { control: 'boolean' },
    paragraph: { control: 'boolean' },
    mark: { control: 'boolean' },
    code: { control: 'boolean' },
  },
}

export const Default = (args) => {
  return (
    <>
      <Text {...args}>Text</Text>
      <Text {...args}>Text</Text>
    </>
  )
}

export const Size = (args) => {
  return (
    <>
      <Text {...args} size="large">
        Text
      </Text>
      <Text {...args} size="normal">
        Text
      </Text>
      <Text {...args} size="small">
        Text
      </Text>
    </>
  )
}
```

- 각 size(string)의 경우를 확인하기 위해 Size를 이름으로 한 Story를 추가로 생성하여 noraml, large, small을 prop으로 전달한 결과를 확인할 수 있도록 작성한다.

학습주제	Search 기능 구현
학습 및 실습 내용	

indexOf()

- 문자열에서 특정 문자의 위치를 구할때 사용한다.
- 특히 사용자가 검색하고 싶은 내용을 검색할때 해당 키워드가 데이터와 일치하는지를 판단할때 많이 사용한다.
- 파라미터로 받은 문자열 혹은 문자가 연속적으로 일치하지 않으면 -1을 return한다.
- 파라미터로 받은 문자열이 연속적으로 일치하면 0을 return한다.
- 파라미터로 받은 문자가 존재하면 해당 index를 return한다.

```
const str = 'abcd'

// 문자 or 문자열
document.write(str.indexOf('w')) // -1
document.write(str.indexOf('abe')) // -1

// 문자열
document.write(str.indexOf('abc')) // 0
document.write(str.indexOf('abcd')) // 0

// 문자
document.write(srt.indexOf('c')) // 2
```

```
{emojis
  .filter((emoji) => emoji.description.indexOf(keyword) >= 0)
  // emoji의 description에 해당하는 문자열에서 사용자가 입력한 keyword가 존재하는 것만
  // filtering 하도록 설정
  .map((emoji, idx) => (
```

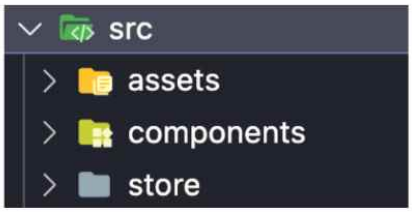
기능 구현하기

```
    <EmojiListItem key={idx} emoji={emoji} />
))}
```

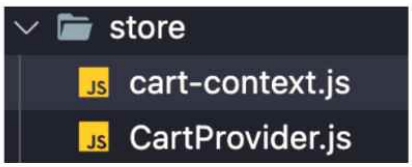
학습주제	React Custom Hooks
학습 및 실습 내용	# useLocalStorage ``` import React, { useState } from "react"; const useLocalStorage = (key, initalValue) => { const [storedValue, setStoredValue] = useState(() => { try { const item = localStorage.getItem(key); return item ? JSON.parse(item) : initalValue; } catch (err) { console.error(err); return initalValue; } }); const setValue = (value) => { try { setStoredValue(value); localStorage.setItem(key, JSON.stringify(value)); } catch (err) { console.error(err); } }; return [storedValue, setValue]; }; export default useLocalStorage; ```

학습주제	React ContextAPI
학습 및 실습 내용	**Context란** • React에서의 Props와 State는 데이터를 다루기 위해 사용되고 Props전달의 흐름은 한 향식이다. • 한쪽에서 흐르는 데이터를 다른 컴포넌트에서 사용하고 싶을 경우 context를 고려할 수 있다. • 즉 전역적인 데이터를 다룰 때 사용한다. • porps를 두 레벨 정도의 컴포넌트를 전달할때(프롭드릴링) 굳이 context를 쓸 필요는 없다. **Context API란** • React에서 전역적인 데이터를 다루기 위해 Flux라는 개념을 도입하고 그에 걸맞는 ContextAPI를 제공하고 있다. • Redux와는 다르게 Context API는 상태관리를 해주는 것이 아니고 상태를 전역적으로 공유해주는 기능만 수행한다. • 실직적인 상태 관리는 useReducer과 useState로 동작하는 것이다. • Context의 Provider와 Consumer를 사용해야한다. • 너무 남용하면 Consumer를 사용하는 컴포넌트들이 모두 렌더링 되므로 성능이 떨어질 수 있다. **ContextAPI 사용법** 1. src폴더 하위에 store폴더를 만든 다.

관습적으로 전체 state를 관리하는 폴더를 store로 쓴다.



2. store폴더 안에 context를 작성한다.
   파일이름은 "**<파일이름>-context.js**"형태로 소문자로 작성한다.



3. react를 import하고 자동 완성을 위해 **createContext**함수를 사용하여 context를 초기화한다.

```
import React from 'react'

const CartContext = React.createContext({
  items: [],
  totalAmount: 0,
  addItem: item => {},
  removeItem: id => {},
})

export default CartContext
```

4. store폴더 안에 **Provider** component를 생성한다.
   생성한 context를 import해서 provider의 value값으로 전달한다.

```
import CartContext from './cart-context'

const CartProvider = ({ children }) => {
  const addItemToCartHander = item => {}

  const removeItemFromCartHandler = id => {}

  const cartContext = {
    items: [],
    totalAmount: 0,
    addItem: addItemToCartHander,
    removeItem: removeItemFromCartHandler,
  }
```

```
  return (
    <CartContext.Provider value={cartContext}>{children}</CartContext.Provider>
  )
}

export default CartProvider
```

5. 생성한 Provider component를 import하고 전달받을 componet들을 wrapping해준다.

```
import CartProvider from './store/CartProvider'

return (
    <CartProvider>
      {cartIsShown && <Cart onClose={hideCartHandler} />}
      <Header onShowCart={showCartHandler} />
      <main>
        <Meals />
      </main>
    </CartProvider>
  )
```

6. context를 사용하고 싶을 때 component에 import하고 useContext함수를 통해 context를 정의한다.

```
import React, { useContext } from 'react'
import CartContext from '../../store/cart-context'

const HeaderCartButton = () => {
  const cartCtx = useContext(CartContext)  // useContext에 context정의

  // context에서의 items를 reduce함수를 통해 총합을 구하는 로직
  const numberOfCartItems = cartCtx.items.reduce((curNumber, item) => {
    return curNumber + items.amount
  }, 0)
}
```