모바일웹 실습일지6-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 학번 | | 201835745 | 이름 | 지서경 |
| 주 의 사 항 | * VS code에서 create-react-app을 이용하되 지정된 프로젝트명으로 작성할 것 - 반드시 react 코드를 사용할 것. * 결과가 맞다면, 브라우져화면을 캡쳐하고, 코드는 텍스트로 입력하여라. * 코드 텍스트의 포맷은 “맑은고딕 9pt”와 “줄간격 110%”으로 통일시킬 것 * **html코드, js코드, jsx 코드, css코드에서 (줄단위로) 주석을 추가하여 코드를 설명하여라.** | | | |
| 실 습 결 과 | 실습1) 강의록6-1의 예제1에 아래조건을 만족하도록 수정하고 완성하여라. 조건1) useEffect hook을 이용하여 (렌더링이 될 때마다) console에 state변수 값을 출력하여라. 조건2) 나머지 조건은 예제1의 동작과정을 모두 포함한다. | | | |
| **<< 실행직후 화면1 캡쳐 (개발자모두 콘솔화면 포함한 화면) >>>**  텍스트, 스크린샷, 모니터이(가) 표시된 사진  자동 생성된 설명 | | | |
| **<< (버튼을 처음 클릭했을 때) 화면2 캡쳐 (개발자모두 콘솔화면 포함한 화면) >>>**  텍스트, 스크린샷, 모니터이(가) 표시된 사진  자동 생성된 설명 | | | |

|  |  |
| --- | --- |
| 실 습 결 과 | **<< (버튼을 5번 클릭했을 때) 화면2 캡쳐 (개발자모두 콘솔화면 포함한 화면) >>>**  텍스트, 모니터, 스크린샷이(가) 표시된 사진  자동 생성된 설명 |
| **<< 리액트 코드 (파일명: toggle.jsx ) 주석포함, 코드넣을 공간이 부족하면 양식을 더 만들어서 추가!** >>  import React, { useState, useEffect } from "react"; //state를 사용을 위한 useState함수와 컴포넌트가 렌더링된 후 작업을 하기 위한 useEffect함수 임포트  //Toggle - 버튼을 클릭할 때마다 버튼 안의 텍스트가 바뀌는 컴포넌트  function Toggle(props) {  const [isToggleOn, setIsToggleOn] = useState(true); //useState함수를 이용해 state변수 isToggleOn 선언 및 초기화  //Toggle 컴포넌트가 렌더링될때마다 실행됨 (즉, 버튼을 클릭하면 state변수가 바뀌면서 재렌더링되기 때문에 버튼을 클릭할때마다 useEffect함수가 실행됨)  useEffect(() => {  //실행직후 handleClick()함수 안에 있는 로그는 보이지 않지만 useEffect() 함수 안에 있는 로그는 보임 (렌더링될때마다 실행되기 때문)  console.log(`useEffect() - state변수(isToggleOn)의 값: ${isToggleOn}`); //isToggleOn의 상태를 콘솔에 출력함 (최신 상태 값이 출력됨)  })  //버튼 클릭시 실행되는 함수  function handleClick() {  setIsToggleOn((isToggleOn) => !isToggleOn); //setIsToggleOn을 통해 state변수인 isToggleOn의 값을 바꿔줌  console.log("토글상태:", isToggleOn); //함수가 실행되었을 때 isToggleOn의 상태를 콘솔에 출력함 (렌더링 되기 전이라 바뀌기 전의 상태 값이 출력됨)  }  return(  <div>  <button onClick={handleClick}> {/\*\* 버튼 클릭시 이벤트핸들러 함수가 실행됨 \*/}  {isToggleOn ? "클릭A" : "클릭B"} {/\*\* isToggleOn의 값이 true면 클릭A, false면 클릭B가 보여짐 \*/}  </button>  </div>  );  }  export default Toggle; //Toggle 컴포넌트 내보냄 |

|  |  |
| --- | --- |
| 실 습 결 과 | 실습2) 강의록6-1의 예제3에 아래조건을 만족하도록 수정하고 완성하여라.  조건1) 이벤트핸들러 handler를 수정하여, 콘솔창에 출력되는 메시지와 똑같은 메시지가 alert창에도 나오도록 하여라.  조건2) 컴포넌트가 렌더링될때는 콘솔창에 “>>> 렌더링되었습니다 !!!” ,으로 출력되도록 한다.  조건3) 컴포넌트가 마운트될떄는 콘솔창에 “>>> 마운트되었습니다 !!!” 으로 출력되도록 한다.  조건4) 컴포넌트가 언마운트될떄는 콘솔창에 “>>> 언마운트되었습니다 !!!” 으로 출력되도록 한다.  조건5) 나머지 조건은 예제3의 동작과정을 모두 포함한다. |
| **<< 실행직후 화면1 캡쳐 (개발자모두 콘솔화면 포함한 화면) >>>**  텍스트이(가) 표시된 사진  자동 생성된 설명 |
| **<< 버튼을 6번 클릭한 직후의 화면캡쳐 (개발자모두 콘솔화면, alert창 모두 포함한 화면) >>>** |

|  |  |
| --- | --- |
| 실 습 결 과 | **<< 리액트 코드 (파일명: lotto.jsx) 주석포함, 코드넣을 공간이 부족하면 양식을 더 만들어서 추가!** >>  //컴포넌트가 렌더링된 후 작업을 하기 위한 useEffect함수, 변수 관리를 위한 useRef함수 임포트  import React, { useEffect, useRef } from "react";  //버튼이 눌릴때마다 당첨회차가 늘어나고 콘솔과 alert창에 회차가 출력되는 컴포넌트  function Lotto() {  const numOfLotto = useRef(1); //useRef함수를 이용해 당첨 회차 변수 numOfLotto 선언 및 초기화 (렌더링 불필요하기 때문에 ref변수 사용)  //useEffect(f)함수를 이용해 컴포넌트가 렌더링될 때마다 콘솔에 "렌더링되었습니다!!!" 출력  useEffect(() => {  console.log("렌더링되었습니다!!!");  });  //useEffect(f,[])함수를 이용해 컴포넌트가 마운트될때는 "마운트되었습니다!!!", 언마운트될때는 "언마운트되었습니다!!!" 출력  useEffect(() => {  console.log("마운트되었습니다!!!");  return() => { console.log("언마운트되었습니다!!!"); };  }, []);  //파라미터가 2개 이상인 이벤트 핸들링 함수  const handler = (str, event) => {  console.log(numOfLotto.current.toString() + "회차 " + str + event.target.value); //콘솔창에 ref변수값(numOfLotto)과 문자열, button의 value 출력  alert(numOfLotto.current.toString() + "회차 " + str + event.target.value); //콘솔창에 출력되는 것과 동일한 내용이 alert창에도 출력됨  numOfLotto.current = numOfLotto.current + 1; //ref변수(당첨 회차) 1씩 증가  }  return(  <div>  {/\*\* 당첨자확인 버튼을 누르면 handler함수 호출 \*/}  <button  value="홍길동"  onClick={(event) => handler("당첨자: ", event)}>{/\*\* 버튼이 눌리면 문자열과 이벤트 객체 두 개의 파라미터가 넘어가면서 함수 호출 \*/}  당첨자 확인  </button>  </div>  );  }  export default Lotto; //Lotto 컴포넌트 반환 |

|  |  |
| --- | --- |
| 실 습 결 과 | 실습3) 강의록6-1의 예제4에 아래조건을 만족하도록 수정하고 완성하여라.  조건1) 로그인 창에 “아이디”를 입력할 수 있도록 input 컴포넌트를 만들어라.  조건2) 만들어야 할 input 컴포넌트는 실시간으로 입력되는 내용이 로그인창 영역(오렌지색)밖의 아래쪽에 출력되도록 하여야 한다.  조건3) LoginButton 컴포넌트 또는 Logout 컴포넌트가 교대로 렌더링될때마다 isLoggedIn 변수와 버튼 클릭 횟수를 콘솔창에 출력되도록 하여라.  조건4) 나머지 조건은 예제4의 동작과정을 모두 포함한다. |
| **<< 실행직후 화면1 캡쳐 (개발자모두 콘솔화면 포함한 화면) >>>** |
| **<< 버튼을 4번 클릭한 직후의 화면캡쳐 (개발자모두 콘솔화면, alert창 모두 포함한 화면) >>>** |

|  |  |
| --- | --- |
| 실 습 결 과 | << 설계한 컴포넌트 구조를 ppt 로 직접 그리고, 캡쳐한 후, 이곳에 삽입한다.  단, 그림속의 글씨크기가 본문의 글씨크기와 동일한 크기로 보여야 한다.>> |
| **<< 리액트 코드 (파일명: input.jsx) 주석포함, 코드넣을 공간이 부족하면 양식을 더 만들어서 추가!** >>  import React from "react"; //React 임포트  //아이디를 입력할 수 있는 input필드 컴포넌트  function Input(props) {  return(  <span>  {/\*\* input필드의 내용이 바뀔때마다(onChange) props를 통해 LoginControl의 onChange함수를 호출함 (입력하는 내용 실시간으로 userId에 업데이트됨) \*/}  아이디: <input id="id" onChange={props.onChange} type="text" className="inputId"></input>&nbsp;  </span>  );  }  export default Input; //Input 컴포넌트 반환 |

|  |  |
| --- | --- |
| 실 습 결 과 | **<< 리액트 코드 (파일명: loginCtrl.jsx ) 주석포함, 코드넣을 공간이 부족하면 양식을 더 만들어서 추가!** >>  import React, { useState, useEffect, useRef } from "react"; //React와 hook함수들 임포트  import Greeting from "./greeting"; //로그인 상태에 따른 출력메세지를 관리하는 컴포넌트 임포트  import LoginButton from "./loginBtn"; //로그인된 상태일때의 로그인 버튼을 반환하는 컴포넌트 임포트  import LogoutButton from "./logoutBtn"; //로그아웃된 상태일때의 로그인 버튼을 반환하는 컴포넌트 임포트  import Input from "./input"; //아이디를 입력받을 수 있는 컴포넌트 임포트  import "./loginCtrl.css"; //스타일 파일 적용  //로그인 관리 컴포넌트  function LoginControl() {  const [isLoggedIn, setIsLoggedIn] = useState(false); //로그인 상태를 담는 state변수 선언 및 초기화  const [userId, setUserId] = useState(""); //Input컴포넌트에서 입력된 내용을 담는 state변수 선언 및 초기화  const btnClickCnt = useRef(0); //버튼 클릭 횟수를 저장하는 ref변수 선언 및 초기화  //로그인된 상태에서 버튼을 눌렀을때의 이벤트 핸들링 함수  const handleLoginClick = () => {  setUserId(""); //로그인되었기 때문에 아이디 입력을 못하므로 p태그에 출력된 입력 내용 클리어  setIsLoggedIn(true); //로그인 상태 true로 변경  btnClickCnt.current = btnClickCnt.current + 1; //버튼 클릭 횟수 1 증가  }  //로그아웃된 상태에서 버튼을 눌렀을때의 이벤트 핸들링 함수  const handleLogoutClick = () => {  setIsLoggedIn(false); //로그인 상태 false로 변경  btnClickCnt.current = btnClickCnt.current + 1; //버튼 클릭 횟수 1 증가  }  //Input에서 받아온 입력내용이 바뀔때마다 실시간으로 userId에 업데이트해주는 함수  const onChange = (e) => {  //id가 "id"인 Input 컴포넌트의 input필드에서 입력된 내용(value)를 받아옴  setUserId((userId) => document.getElementById("id").value);  };  //의존성변수 isLoggedIn의 값이 바뀔때마다(LoginButton, LogoutButton 컴포넌트가 교대로 렌더링될때마다) 호출되는 이펙트 함수  useEffect(() => {  //현재 isLoggedIn 변수의 상태와 버튼 클릭 횟수가 콘솔창에 출력됨  console.log(`[useEffect] isLoggedIn: ${isLoggedIn}, 버튼 클릭 횟수: ${btnClickCnt.current}`);  }, [isLoggedIn]); //의존성배열[isLoggedIn]  let button; //버튼 컴포넌트를 담을 변수  let inputId; //Input 컴포넌트를 담을 변수  //로그인 상태에 따라 변수에 저장되는 컴포넌트가 다름  if(isLoggedIn) {  //로그인된 상태라면 LogoutButton 컴포넌트를 렌더링  button = <LogoutButton onClick={handleLogoutClick} />;  }  else {  //로그아웃된 상태라면 Input 컴포넌트와 LoginButton 컴포넌트를 렌더링  inputId = <Input onChange={onChange} />  button = <LoginButton onClick={handleLoginClick} />;  }  return(  <div>  <div className="login">{/\*\* 로그인 영역(주황색 박스) \*/}  <div className="greeting">{/\*\* 출력메세지 영역 \*/}  <Greeting isLoggedIn={isLoggedIn} />{/\*\* 로그인 상태에 따라 다른 메세지가 출력됨 \*/}  </div>  <div className="button">{/\*\* 로그인버튼 영역 \*/}  {inputId}{/\*\* 로그아웃된 상태에서만 보이는 아이디 입력 필드 \*/}  {button}{/\*\* 로그인 상태에 따라 다른 버튼이 보여짐 \*/}  </div>  </div>  <div className="idTxt">{/\*\* 아이디 출력 영역(로그아웃된 상태에서만 보임) \*/}  <p id="idTxt">{userId}</p>{/\*\* 입력하고 있는 아이디를 실시간으로 보여줌 \*/}  </div>  </div>  );  }  export default LoginControl; //LoginControl 컴포넌트 반환 |