차례

머리말<저자 요청> 머리말 원고 요망

리액트 개발 환경 준비하기

1부 입문: <저자 요청> 제목 요망

001. jsx에 html 적용하기

002. jsx에 css 적용하기

003. Component 사용하기

004. 생명주기 함수 render() 사용하기

005. 생명주기 함수 constructor(props) 사용하기

006. 생명주기 함수 static getDerivedStateFormProps(props, state) 사용하기

007. 생명주기 함수 componentDidMount() 사용하기

008. 생명주기 함수 shouldComponedntUpdate() 사용하기

009. 템플릿 문자열 사용하기

010. var, let, const 사용하기

011. 전개 연산자 사용하기

012. class 사용하기

013. 화살표 함수 사용

014. forEach() 함수 사용하기

015. map() 함수 사용하기

016. jquery 사용하기

2부 초급: React.js 기초 다지기

017. props 사용하기

018. props 자료형 선언하기

019. props Boolean으로 사용하기

020. props 객체형으로 사용하기

021. props 필수 값으로 사용하기

022. props 기본 값 정의하기

023. props 자식 Component에 node전달하기

024. state 사용하기

025. setState() 함수 사용하기

026. state를 직접 변경한 후 forceUpdate() 함수 사용하기

027. Component(class형 컴포넌트)

028. PureComponent(class형 컴포넌트)

029. shallow-equal 사용하기(class형 컴포넌트)

030. 함수형 컴포넌트 사용하기

031. Hook 사용하기

032. Fragments 사용하기

033. map()으로 element 반환하기

034. reactstrap Alerts 사용하기

035. reactstrap Badge 사용하기

036. reactstrap Breadcrumbs 사용하기

037. reactstrap Button Dropdown 사용하기

038. reactstrap Button Group 사용하기

039. reactstrap Bottons 사용하기

040. reactstrap Card 사용하기

041. reactstrap Carousel 사용하기

042. reactstrap Collapse 사용하기

043. reactstrap Fade 사용하기

044. reactstrap Form 사용하기

045. reactstrap Input Group 사용하기

046. reactstrap Jumbotron 사용하기

047. reactstrap List Group 사용하기

048. reactstrap Modal 사용하기

049. reactstrap Navbar 사용하기

050. reactstrap Pagination 사용하기

051. reactstrap Popovers 사용하기

052. reactstrap Progress 사용하기

053. reactstrap Spinner사용하기

054. reactstrap Table 사용하기

055. reactstrap Tab 사용하기

056. sweetalert2 Basic 사용하기

057. sweetalert2 position 사용하기

058. sweetalert2 confirm 사용하기

59. fetch get 호출하기

60. fetch post 호출하기

61. aixos get 사용하기

62. aixos post 사용하기

63. 콜백 함수 사용하기

64. Promise then 사용하기

65. Promise catch 사용하기

3부 중급: <저자 요청> 제목 요망

66. click 이벤트 사용하기(onClick)

67. change 이벤트 사용하기(onChange)

68. mousemove 이벤트 사용하기(onMouseMove)

69. mouseover 이벤트 사용하기(onMouseOver)

70. mouseout 이벤트 사용하기(onMouseOut)

71. key 이벤트 사용하기(onKeyDown, onKeyPress, onKeyUp)

72. submit 이벤트 사용하기(onSubmit)

73. Ref 사용하기

74. 커링 함수 구현하기

75. 하이오더 컴포넌트 구현하기

76. 컨텍스트 API 사용하기

77. 컨텍스트로 부모 데이터 변경하기

78. redux 리듀서로 스토어 생성

79. redux 스토어 상속과 디스패치 함수 실행

80. redux 리듀서에서 스토어 데이터 변경

81. react-redux로 스토어 상속

82. react-redux로 스토어 데이터 사용

83. react-redux로 스토어 데이터 변경

84. redux 미들웨어 사용하기

85. react-cookies save 사용하기

86. react-cookies load 사용하기

87. react-cookies remove 사용하기

88. react-router-dom BrowserRouter 사용하기

89. react-router-dom Route 사용하기

90. react-router-dom Link 사용하기

91. header, footer 구현하기

92. home 페이지 만들기

93. <react img> 태그 사용하기

94. lodash 디바운스 사용하기

95. lodash 스로틀 사용하기

4부 활용: <저자 요청> 제목 요망

96. 서울시 유동 인구 데이터 사용하기 1 - 리스트 페이지 만들기

97. 서울시 유동 인구 데이터 사용하기 2 - Open Api Key 발급 받기

98. 서울시 유동 인구 데이터 사용하기 3 - JSON Viewer로 데이터 확인하기

99. 서울시 유동 인구 데이터 사용하기 4 - axios로 Api 호출하기

100. recharts로 LineChart 구현하기 1

101. recharts로 LineChart 구현하기 2(서울시 유동 인구 데이터)

102. recharts로 AreaChart 구현하기(서울시 유동 인구 데이터)

103. recharts로 BarChart 구현하기(서울시 유동 인구 데이터)

104. recharts로 ComposedChart 구현하기(서울시 유동 인구 데이터)

105. recharts로 ScatterChart 구현하기(서울시 유동 인구 데이터)

5부 실무: <저자 요청> 제목 요망

106. node.js express 프레임워크 설치

107. node 서버 구동<저자 요청> 누락 원고 요망

108. node 서버 api 호출하기

109. react 서버와 node 서버를 프록시로 연결하기

110. node서버 Api를 get으로 json 데이터 호출하기

111. node서버 Api를 post로 json 데이터 호출하기

112. concurrently로 react, node 서버 한 번에 구동하기

113. MYSQL 서버 구축하기 1 – AWS 가입하기

114. MYSQL 서버 구축하기 2 – RDS 인스턴스 생성하기

115. MYSQL 서버 구축하기 2 – RDS 인스턴스 포트 설정하기

116. MYSQL client 설치하기

117. workbench로 rds 서버 접속하기

118. MYSQL 스키마 생성 및 table 생성하기

119. MYSQL 한국 시간 설정하기

120. MYSQL 한글 설정 및 Safe 모드 해제

121. MYSQL 더미 데이터 삽입하기

122. NODE 조회 Api 만들기 1 - body-parser 패키지 사용하기

123. NODE 조회 Api 만들기 2 – mysql 서버 연결하기

124. NODE 조회 Api 만들기 3 – mybatis 패키지 사용하기

125. NODE 조회 Api 만들기 4 – Api 호출로 데이터 확인하기

126. REACT 조회 페이지 만들기

127. DB 연결 풀 구현하기

128. NODE 등록 Api 만들기 1 - insert 라우터 분기

129. NODE 등록 Api 만들기 2 - insert 쿼리 추가, response 처리

130. REACT 등록 페이지 만들기

131. REACT 상세 조회 페이지 만들기

132. NODE 상세 조회 select 쿼리 where절 추가

133. NODE 수정 Api 만들기 1 - update 라우터 분기

134. NODE 수정 Api 만들기 2 - update 쿼리 추가, response 처리

135. REACT 수정 페이지 만들기

136. NODE 삭제 Api 만들기 1 - delete 라우터 분기

137. NODE 삭제 Api 만들기 2 - delete 쿼리 추가, response 처리

138. REACT 리스트 페이지 삭제 기능 구현

139. 파일, 이미지 업로드 Api 만들기 – multer 패키지 사용

140. REACT 등록 페이지 - 파일, 이미지 업로드 Api 호출하기

141, REACT 등록 페이지 - 업로드 경로를 db에 insert하기

142. REACT 상세 조회 페이지 - 파일, 이미지명 노출하기

143. REACT 수정 페이지 - 파일, 이미지 업로드 Api 호출하기

144. REACT 수정 페이지 - 업로드 경로 db에 update하기

145. 회원 정보 table 생성하기

146. NODE 회원 가입 Api 만들기 1 - bcrypt로 암호화하기

147. NODE 회원 가입 Api만들기 2 - insert 쿼리 추가

148. REACT 회원 가입 페이지 만들기 1 - 입력 form 만들기

149. REACT 회원 가입 페이지 만들기 2 - 유효성 체크하기

150. REACT 회원 가입 페이지 만들기 3 - 아이디 중복 체크하기

151. REACT 회원 가입 페이지 만들기 3 – 회원 가입 api 호출하기

152. NODE 로그인 Api 만들기 1 - 라우터 분기, 쿼리 추가

153. NODE 로그인 Api 만들기 2 - bcrypt로 비밀번호 비교하기

154. REACT 로그인 페이지 만들기 - 로그인 api 호출하기

155. 회원 정보 암호화 Api 만들기 - jwt로 회원 정보 암호화하기

156. REACT 쿠키로 로그인 유지하기 - 쿠키에 회원 정보 저장하기

157. 회원 정보 복호화 Api 만들기 - jwt로 회원 정보 복호화하기

158. 로그인 회원에게 권한 부여하기 - 쿠키에서 회원 정보 불러오기

159. 로그인 세션 시간 관리 - 쿠키 유효 시간 연장하기

160. 로그인 상태에서만 header 노출하기, 로그아웃 구현하기

161. 비밀번호 재설정 메일 템플릿 만들기

162. 메일 발송 Api 만들기 – nodemailer 패키지 사용

163. 비밀번호 조회 Api 만들기 - 라우터 분기, 쿼리 추가

164. 아이디와 회원 이름으로 비밀번호 조회 Api 호출하기

165. 비밀번호 재설정 이메일 발송 Api 호출하기

166. GMAIL 계정 접근 권한 허용하기

167. 수신된 이메일과 버튼 링크 확인하기

168. 이메일 토큰으로 사용자 인증하기

169. 비밀번호 수정 Api 만들기 - 라우터 분기, 쿼리 추가

170. 비밀번호 수정 Api 호출하기

171. 스케줄러 만들기 1 - 배치 테이블 생성하기

172. 스케줄러 만들기 2 - node-cron 패키지 사용

173. 스케줄러 만들기 3 - 특정 시간에 insert 쿼리 실행하기

174. 웹 서버 구축하기 1 - AWS EC2 인스턴스 생성

175. 웹 서버 구축하기 2 - 포트 설정, 키 페어 다운로드

176. 웹 서버 구축하기 3 - 탄력적 IP 설정하기

177. MobaXterm - SFTP로 EC2 서버 접속하기

178. MobaXterm - SSH로 EC2 서버 접속하기

179. SFTP로 EC2 서버에 로컬 소스 옮기기

180. SSH로 소스 압축 풀기, EC2 서버에 nodejs 설치하기

181. react, node 서버에 필요한 패키지 설치하기

182. react, node 서버 구동해 외부에서 접속하기

183. EC2 서버 시간 한국으로 변경하기

184. vi 에디터로 EC2 서버 소스 직접 수정하기

185. chown, chmod 명령어로 서버 접근 권한 부여하기

186. nohup으로 서버 상시 구동하기, 포트로 서버 종료하기

187. kakao 지도 API appkey 발급, 사이트 도메인 등록하기

188. react 서버 포트 80으로 설정하기

189. react에서 kakao 지도 생성하기

190. 코로나 확진자 정보 excel 파일을 json 파일로 변환하기

191. 확진자 위치에 마커 추가하기

192. 확진자 위치에 이미지 마커 추가하기

193. 마커에 클러스터러 사용하기

194. 확진자 정보 커스텀 오버레이로 노출하기

195. EC2 서버에 로컬 react소스 옮기기

196. EC2 서버에 react패키지 설치하기

197. EC2 서버에 80포트 설정하기

198. 카카오 사이트 도메인에 EC2 IP 추가하기

199. screen으로 서버 상시 구동하기, 포트로 서버 종료하기

200. 도메인 구매 후 ip 연결하기

머리말

<저자 요청: 1페이지 분량의 원고 부탁드립니다.>

(앞부속)리액트 개발 환경 준비하기

개발 환경을 준비하기 전에 설치 도구에 대한 정리가 필요하다. 자바스크립트 패키지 관리 도구에는 npm와 yarn이 있다. 먼저 npm은 ‘node package manager’의 약자로, node.js와 react.js에서 사용하는 대부분의 패키지를 설치할 수 있다. npm의 경우 node.js와 함께 설치된다. yarn은 facebook에서 만든 패키지 관리 도구로, npm에 비해 캐싱, 보안, 신뢰성 등이 개선됐다. 이 책에서는 1~4부의 react 예제에 활용되는 패키지를 yarn으로 설치하고 npm은 node.js 실습을 하는 5부에서 사용한다.

**1. node.js 다운로드**nodejs공식 웹 사이트([https://nodejs.org/en/download/releases](https://nodejs.org/en/download/releases/))에 접속해 14.4.0 버전을 다운로드한다. 최신 버전을 사용해도 괜찮지만 이후에 나올 예제에서 14.4.0 버전을 사용했을 때 문제가 없었기 때문에 14.4.0 버전의 사용을 권장한다. [Downloads]를 클릭하면 다운로드할 파일을 선택할 수 있다. 64비트 윈도우 운영체제인 경우에는 node-v14.4.0-x64.msi를 다운로드한다.

|  |
| --- |
|  |

**<그림 1>**

**node.js 공식 홈 페이지**

**2. node.js 설치**

다운로드한 파일을 실행해 설치를 시작한다.

|  |
| --- |
|  |

**<그림 2>**

설치 도중 다음 화면에서는 [Automatically install the necessary tools. Note that this will also install Boxstarter and Chocolatey. The script will pop-up in a new window after the installation completes]에 체크 표시를 하지 않고 [Next]를 클릭한다.

|  |
| --- |
|  |

**<그림 3>**

설치를 완료한다.

|  |
| --- |
|  |

**<그림 4>**

**3. node.js 및 npm 설치 확인**

[윈도우 키] + [R]을 누른 후 ‘cmd'를 입력해 명령 프롬프트를 실행한다. node -v, npm -v 명령어로 설치된 node.js와 npm 버전을 확인한다.

|  |
| --- |
|  |

**<그림 5>**

**4. react 프로젝트의 workspace로 사용할 폴더 생성**

cmd 창에서 cd 명령어를 이용해 생성한 폴더 경로로 이동한다.

|  |
| --- |
|  |

**<그림 6>**

**5. yarn 설치**

npm install 명령어로 yarn을 설치한다. yarn -v 명령어로 설치된 yarn 버전을 확인한다.

|  |
| --- |
|  |

**<그림 7>**

**6. create-react-app 설치**

npm install -g create-react-app 명령어로 create-react-app을 설치한다. create-react-app client 명령어로 ‘client’라는 프로젝트를 생성한다.

|  |
| --- |
|  |

**<그림 8>**

프로젝트를 생성한 후 client 경로를 보면 package.json, node\_modules 등의 파일, 폴더가 생성된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 9>**

**7. react 서버 실행**

Client 경로에서 yarn start 명령어로 react 서버를 실행한다.

|  |
| --- |
|  |

**<그림10>**

크롬 웹 브라우저에서 http://localhost:3000 URL을 열면 다음과 같은 화면을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 11>**

**9. react 서버 실행**

src 폴더의 index.js 파일을 열어 <React.StrictMode>, </React.StrictMode> 태그를 삭제한다. Strict 모드는 애플리케이션 내의 잠재적 문제를 알아내기 위한 도구로, 나중에 배울 생명주기 함수를 여러 번 실행하는 원인이 되므로 사용하지 않는다.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15 | import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(  <App />,  document.getElementById('root')  );  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

**9. 예제 파일을 실행하는 방법**

react 프로젝트인 경우, react 경로에서 [npm install] 명령어를 실행한다. 같은 경로에서 [npm start] 명령어로 react 서버를 구동한다. react, node 프로젝트인 경우, react와 node 경로에서 각각 [npm install] 명령어를 실행한 후 [yarn dev] 명령어로 react, node 서버를 동시에 구동한다. react, node 폴더 경로에는 각각 package.json 파일이 있는데, 프로젝트에서 사용하는 npm 패키지 리스트가 작성돼 있다. [npm install] 또는 [yarn install]을 실행하면 package.json 파일에 있는 패키지 리스트가 설치된다.

(도비라)part1 입문: <저자 요청> 제목 요망(2부 도비라 참조)

**<저자 요청: 5~6줄 정도의 발문 부탁드립니다.>**

001. jsx에 html 적용하기

**• 학습 내용:** react에서 html 코드를 적용하는 방법을 이해한다.

**• 힌트 내용:** jsx 소스에서 return() 안에 html 코드를 입력한다.

create-react-app 템플릿으로 프로젝트를 생성하면, App.js의 html 코드를 이용해 홈 화면을 그려준다. 여기서 홈 화면이란 localhost:3000(도메인) 또는 127.0.0.1:3000(IP)와 같이 페이지 상세 URL이 뒤에 붙지 않은 웹 사이트의 대표 페이지를 말한다.

create-react-app 템플릿의 홈 화면인 App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12 | import React from 'react';  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>HTML 적용하기</p>  </div>  );  }  export default App; |

5~8: 스타일이 적용되지 않은, 기본 HTML 코드만 삽입된 상태다. 웹 브라우저에서 페이지를 열면, 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 12>**

002. jsx에 css 적용하기

**• 학습 내용:** react에서 css 파일로 html 코드에 스타일을 적용하는 방법을 이해한다.

**• 힌트 내용:** css 파일을 별도로 만든 후 jsx 파일에서 임포트(import)해 사용한다.

css란, html과 같은 문서에 스타일을 적용할 때 사용하는 언어다. css를 적용하는 방법은 크게 css 코드를 html 코드 내부에 삽입하는 방법, html 코드가 있는 파일에 함께 정리하는 방법, css 코드만 따로 파일로 작성하는 방법으로 나눌 수 있다. 이번 예제에서는 세 번째 방법을 적용한다.

App.css 파일을 다음과 같이 수정한다.

<소스> App.css

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14 | div {  background-color: rgb(162, 127, 243);  color: rgb(255, 255, 255);  padding: 40px;  font-family: 고딕;  text-align: center;  }  h1 {  color: white;  background-color: #2EFE2E;  padding: 10px;  font-family: 궁서;  } |

1~7: <div> 태그의 속성 중 background-color는 배경색, color는 글자색, padding은 태그 안쪽 여백, font-family는 글자 폰트, text-align은 글자 정렬 값을 지정했다.

9~14: <h1> 태그의 속성 중 color는 글자색, background-color는 배경색, padding은 태그 안쪽 여백, font-family는 글자 폰트 값을 지정했다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13 | import React from 'react';  import './App.css';  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  </div>  );  }  export default App; |

2: App.js와 같은 폴더 경로에 위치한 App.css를 임포트한다. App.css가 App.js보다 한 단계 상위 폴더에 위치한다면, import './../App.css'와 같이 사용해 상위 경로에 접근해야 한다.

6~9: App.css에서 작성돼 있는 <div>와 <h1> 태그에 스타일이 적용된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 13>**

003. Component 사용하기

**• 학습 내용:** react에서 component를 사용해 다른 파일에 있는 html 코드를 이식해 사용하는 방법을 이해한다.

**• 힌트 내용:** 다른 파일에서 작성한 html 코드를 component 단위로 임포트해 사용한다.

component란, 특정 코드 뭉치를 다른 부분에 이식하거나 재사용하기 위해 사용하는 코드 블록 단위를 말한다. component를 파일 단위로 작성해 필요한 위치에서 임포트해 사용할 수 있다.

src 폴더에 R003\_ImportConponent.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R003\_ImportConponent.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11 | import React, { Component } from 'react';  class R003\_ImportConponent extends Component {  render () {  return (  <h2>[THIS IS IMPORTED COMPONENT ]</h2>  )  }  }  export default R003\_ImportConponent; |

3: 첫 번째 줄에서 임포트한 component를 상속받아 R003\_ImportConponent 클래스에서 사용한다.

4~6: return된 html 코드를 render() 함수를 사용해 화면에 표시한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14 | import React from 'react';  import './App.css';  import ImportConponent from './R003\_ImportConponent'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ImportConponent></ImportConponent>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R003\_ImportConponent.js 파일을 임포트해 사용할 수 있도록 한다.

10: R003\_ImportConponent.js에서 작성한 component를 사용한다. 이때 render() 함수에서 읽어 들였던 '<h2>[THIS IS IMPORTED COMPONENT ]</h2>' 코드가 이식된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 14>**

004. 생명주기 함수 render() 사용하기

**• 학습 내용:** 컴포넌트의 생명주기 함수 중 render() 함수에 대해 이해한다.

**• 힌트 내용:** render() 함수가 실행될 때 로그를 출력하고 콘솔에서 로그를 확인한다.

react에서 생명주기란 component의 생성, 변경, 소멸 과정을 뜻한다. 4~7번 예제에서 학습하는 constructor(), render(), getDerivedStateFormProps(), componentDidMount() 함수들은 component의 '생성' 과정에 속하는 함수다.

src 폴더에 R004\_LifecycleEx.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R004\_LifecycleEx.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12 | import React, { Component } from 'react';  class R004\_LifecycleEx extends Component {  render() {  console.log('3. render Call');  return (  <h2>[THIS IS RENDER FUCNTION ]</h2>  )  }  }  export default R004\_LifecycleEx; |

4~9: render()는 return되는 html 형식의 코드를 화면에 그려주는 함수다. 화면 내용이 변경돼야 할 시점에 자동으로 호출된다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import LifecycleEx from './R004\_LifecycleEx'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <LifecycleEx></LifecycleEx>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R004\_LifecycleEx.js 파일을 임포트해 사용할 수 있도록 한다.

10: R004\_LifecycleEx.js에서 작성한 component를 이식한다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 render() 함수에서 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 15>**

005. 생명주기 함수 constructor(props) 사용하기

**• 학습 내용:** 컴포넌트의 생명주기 함수 중 constructor() 함수에 대해 이해한다.

**• 힌트 내용:** 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.

src 폴더에 R005\_LifecycleEx.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R005\_LifecycleEx.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18 | import React, { Component } from 'react';  class R005\_LifecycleEx extends Component {  constructor(props) {  super(props);  this.state = {};  console.log('1. constructor Call');  }  render() {  console.log('3. render Call');  return (  <h2>[THIS IS CONSTRUCTOR FUCNTION ]</h2>  )  }  }  export default R005\_LifecycleEx; |

4~6: constructor(props) 함수는 생명주기 함수 중 가장 먼저 실행되며 처음 한 번만 호출된다. component 내부에서 사용되는 변수(state)를 선언하고 부모 객체에서 전달받은 변수(props)를 초기화할 때 사용한다. super() 함수는 가장 위에 호출해야 한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import LifecycleEx from './R005\_LifecycleEx'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <LifecycleEx></LifecycleEx>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R005LifecycleEx.js 파일을 임포트해 사용할 수 있도록 한다.

10: R005LifecycleEx.js에서 작성한 component를 이식한다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 render() 함수에서 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 16>**

006. 생명주기 함수 static getDerivedStateFormProps(props, state) 사용하기

**• 학습 내용:** 컴포넌트의 생명주기 함수 중 getDerivedStateFormProps() 함수에 대해 이해한다.

**• 힌트 내용:** 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17 | import React from 'react';  import './App.css';  import LifecycleEx from './R006\_LifecycleEx'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <LifecycleEx  prop\_value = 'FromApp.js'  />  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R006\_LifecycleEx.js 파일을 임포트해 사용할 수 있도록 한다.

10~12: App.js에서 임포트한 component인 R006\_LifecycleEx로 prop\_value라는 변수를 전달한다.

src 폴더에 R006\_LifecycleEx.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R006\_LifecycleEx.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  class R006\_LifecycleEx extends Component {  static getDerivedStateFromProps(props, state) {  console.log('2. getDerivedStateFromProps Call :'+props.prop\_value);  return {};  }  constructor(props) {  super(props);  this.state = {};  console.log('1. constructor Call');  }  render() {  console.log('3. render Call');  return (  <h2>[THIS IS CONSTRUCTOR FUCNTION ]</h2>  )  }  }  export default R006\_LifecycleEx; |

4~6: getDerivedStateFromProps(props, state) 함수는 constructor() 함수 다음으로 실행된다. 컴포넌트가 새로운 props를 받게됐을 때 state을 변경해준다. App.js에서 전달한 prop\_value라는 변수를 props.prop\_value로 접근해 값을 가져올 수 있다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 17>**

007. 생명주기 함수 componentDidMount() 사용하기

**• 학습 내용:** 컴포넌트의 생명주기 함수 중 componentDidMount() 함수에 대해 이해한다.

**• 힌트 내용:** 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.

App.js 파일에서 LifecycleEx를 임포트하는 부분을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3 | ..  import LifecycleEx from './R007\_LifecycleEx'  .. |

3: App.js 파일과 같은 경로에 위치한 R007\_LifecycleEx.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R007\_LifecycleEx.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R007\_LifecycleEx.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, { Component } from 'react';  class R007\_LifecycleEx extends Component {  static getDerivedStateFromProps(props, state) {  console.log('2. getDerivedStateFromProps Call :'+props.prop\_value);  return {tmp\_state:props.prop\_value};  }  constructor(props) {  super(props);  this.state = {};  console.log('1. constructor Call');  }  componentDidMount() {  console.log('4. componentDidMount Call');  console.log('5. tmp\_state : '+this.state.tmp\_state);  }  render() {  console.log('3. render Call');  return (  <h2>[THIS IS COMPONENTDIDMOUNT FUCNTION ]</h2>  )  }  }  export default R007\_LifecycleEx; |

4~6: componentDidMount() 함수는 작성한 함수들 중 가장 마지막으로 실행된다. render() 함수가 return되는 html 형식의 코드를 화면에 그려준 후 실행된다. 화면이 모두 그려진 후에 실행돼야 하는 이벤트 처리, 초기화 등 가장 많이 활용되는 함수다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 18>**

008. 생명주기 함수 shouldComponedntUpdate() 사용하기

**• 학습 내용:** 컴포넌트의 생명주기 함수 중 shouldComponedntUpdate() 함수에 대해 이해한다.

**• 힌트 내용:** 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.

react에서 생명주기란 component의 생성, 변경, 소멸의 과정을 뜻한다. shouldComponedntUpdate() 함수는 component의 '변경' 과정에 속하는 함수다. 여기서 '변경'이란, props나 state의 변경을 말한다.

App.js 파일에서 LifecycleEx를 임포트하는 부분을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3 | ..  import LifecycleEx from './R008\_LifecycleEx'  .. |

2: App.js 파일과 같은 경로에 위치한 R008\_LifecycleEx.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R008\_LifecycleEx.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R008\_LifecycleEx.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import React, { Component } from 'react';  class R008\_LifecycleEx extends Component {  static getDerivedStateFromProps(props, state) {  console.log('2. getDerivedStateFromProps Call :'+props.prop\_value);  return {tmp\_state:props.prop\_value};  }  constructor(props) {  super(props);  this.state = {};  console.log('1. constructor Call');  }  componentDidMount() {  console.log('4. componentDidMount Call');  console.log('5. tmp\_state : '+this.state.tmp\_state);  this.setState({tmp\_state2 : true});  }  shouldComponentUpdate(props, state) {  console.log('6. shouldComponentUpdate Call / tmp\_state2 = ' + state.tmp\_state2);  return state.tmp\_state2  }  render() {  console.log('3. render Call');  return (  <h2>[THIS IS shouldComponentUpdate FUCNTION ]</h2>  )  }  }  export default R008\_LifecycleEx; |

15: componentDidMount() 함수는 '생성' 단계의 생명주기 함수 중 가장 마지막으로 실행된다.

18: tmp\_state2라는 state 변수에 true라는 boolean 타입의 데이터를 세팅했다. setState() 함수는 변수의 선언과 초기화를 동시에 실행한다.

21~23: line 18에서 state의 변경이 발생했기 때문에 '변경' 단계의 생명주기 함수 shouldComponentUpdate()가 실행된다. shouldComponentUpdate()는 boolean 타입의 데이터를 반환하는데, return 값이 true일 경우에 render() 함수를 한 번 더 호출한다.

|  |
| --- |
| NOTE |
| shouldComponentUpdate() 함수의 반환 값에 따라 render() 함수를 재실행할 수 있다는 점을 이용하면, props나 state 변수가 변경될 때 화면을 다시 그리며 제어할 수 있다. |

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 19>**

009. 템플릿 문자열 사용하기

**• 학습 내용:** ES6 문자열 사용 방법에 대해 이해한다.

**• 힌트 내용:** 기존 자바스크립트 문자열과 비교하고 추가된 함수를 확인한다.

ES(ECMA스크립트)는 표준화된 스크립트 언어인데, ES 뒤에 붙은 숫자는 버전을 의미한다. 2011년부터 사용된 ES5가 웹 표준처럼 사용되고 있다. 2015년 발행된 ES6는 많은 유용한 기능이 추가됐고 자바스크립트는 이 기술 규격을 따른다. react도 자바스크립트 기반의 언어이기 때문에 ES6의 모든 기능을 사용할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Es6 from './R009\_Es6'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Es6/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R009\_Es6.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R009\_Es6.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R009\_Es6.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import React, { Component } from 'react';  class R009\_Es6 extends Component {  constructor(props) {  super(props);  this.state = {};  }  componentDidMount() {  var jsString1 = '자바스크립트'  var jsString2 = '입니다\n다음 줄입니다.'  console.log(jsString1+' 문자열'+jsString2+'~');    var Es6String1 = 'ES6'  var Es6String2 = '입니다'  console.log(`${Es6String1} 문자열${Es6String2}!!  \_\_\_\_다음줄입니다`);  var LongString = "ES6에 추가된 String 함수들입니다.";  console.log('startsWith : '+LongString.startsWith("ES6에 추가"));  console.log('endsWith : '+LongString.endsWith(" 함수들입니다."));  console.log('includes : '+LongString.includes("추가된 String"));  }  render() {  return (  <h2>[THIS IS ES6 STRING ]</h2>  )  }  }  export default R009\_Es6; |

12: 기존 자바스크립트에서는 줄바꿈을 하려면 개행 문자(＼n)를 사용해야 한다.

13: 문자열과 변수를 합치기 위해서는 작은 따옴표(또는 큰따옴표)로 문자열을 감싸고 +로 연결해야 한다.

17~18: 따옴표가 아닌 백틱(`)으로 전체 문자열과 변수를 묶어 사용한다. 변수는 ${ 변수명} 형태로 넣고 코드상에서 줄바꿈을 하면 개행 문자 없이도 사용할 수 있다.

21~23: startsWith(), endsWith(), includes()는 ES6에 추가된 String 함수들이다. startsWith()는 변수 앞에서 부터, endsWith()는 뒤에서부터 일치하는 문자열이 있는지 찾는다. includes()는 위치에 상관없이 변수에 특정 문자열이 포함돼 있는지 판단한다. 함수 조건에 부합하면 true, 부합하지 않으면 false를 반환한다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 20>**

010. var, let, const 사용하기

**• 학습 내용:** ES6의 변수 선언 방식인 let과 const에 대해 이해한다.

**• 힌트 내용:** 기존 var 변수와 비교해 변수의 재선언, 재할당이 가능한지 확인한다.

ES5에서 사용하던 var는 유연한 방식으로 변수를 재선언, 재할당할 수 있다. 이런 특징으로 인해 변수의 사용 범위가 불확실해지거나 의도하지 않은 변숫값 변경이 발생할 수 있다. 이러한 var의 단점을 보완하기 위에 ES6에서 let과 const가 추가됐다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Variable from './R010\_Variable'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Variable/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R010\_Variable.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R010\_Variable.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R010\_Variable.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | import React, { Component } from 'react';  class R010\_Variable extends Component {  constructor(props) {  super(props);  this.state = {};  }  componentDidMount() {  var varName = 'react';  console.log('varName1 : '+varName);  var varName = '200'; // 'varName' is already defined no-redeclare  console.log('varName2 : '+varName);  let letName = 'react'  console.log('letName1 : '+letName);  // let letName = '200' // Parsing error: Identifier 'letName' has already been declared  letName = 'react200';  console.log('letName2 : '+letName);  const constName = 'react';  console.log('constName : ' + constName);  // const constName = '200' // Parsing error: Identifier 'constName' has already been declared  // constName = 'react200' // Uncaught TypeError: Assignment to constant variable.  }  render() {  return (  <h2>[THIS IS Variable ]</h2>  )  }  }  export default R010\_Variable; |

13: 이미 선언한 var 변수 varName를 다시 선언했을 때 'varName' is already defined no-redeclare라는 경고 메시지가 콘솔 로그에 출력된다. 하지만 var 변수는 재선언, 재할당을 허용하기 때문에 경고 메시지가 출력돼도 페이지가 정상적으로 표시된다.

18: 이미 선언한 let 변수 letName를 다시 선언했을 때 Parsing error: Identifier 'letName' has already been declared라는 에러 메시지가 콘솔 로그에 출력된다. let 변수는 재선언을 허용하지 않기 때문에 에러 페이지가 표시된다.

19: let 변수는 재할당을 허용한다. 이미 선언한 let 변수 letName에 새로운 값을 할당했을 때 페이지가 정상적으로 표시된다.

24: 이미 선언한 const 변수 constName를 다시 선언했을 때 Parsing error: Identifier 'constName' has already been declared라는 에러 메시지가 콘솔 로그에 출력된다. const 변수는 재선언을 허용하지 않기 때문에 에러 페이지가 표시된다.

25: 이미 선언한 const 변수 constName에 새로운 값을 할당했을 때 Uncaught TypeError: Assignment to constant variable라는 에러 메시지가 콘솔 로그에 출력된다. const 변수는 재할당을 허용하지 않기 때문에 에러 페이지가 표시된다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 21>**

011. 전개 연산자 사용하기

**• 학습 내용:** 전개 연산자를 통해 배열과 객체의 데이터를 합치거나 추출하는 방법을 이해한다.

**• 힌트 내용:** 기존 ES5 문법과 비교해 ES6 전개 연산자의 장점과 사용법을 확인한다.

전개 연산자는 배열이나 객체 변수를 좀 더 직관적이고 편리하게 합치거나 추출할 수 있게 도와주는 문법이다. 변수 앞에 …(마침표 3개)를 입력해 사용한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import SpreadOperator from './R011\_SpreadOperator'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <SpreadOperator/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R011\_SpreadOperator.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R011\_SpreadOperator.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R011\_SpreadOperator.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | import React, { Component } from 'react';  class R011\_SpreadOperator extends Component {  constructor(props) {  super(props);  this.state = {};  }  componentDidMount() {  //javascript Array  var varArray1 = ['num1', 'num2'];  var varArray2 = ['num3', 'num4'];  var sumVarArr = [varArray1[0], varArray1[1], varArray2[0], varArray2[1]];  // var sumVarArr = [].concat(varArray1, varArray2);  console.log('1. sumVarArr : '+sumVarArr);  //ES6 Array  let sumLetArr = [...varArray1, ...varArray2];  console.log('2. sumLetArr : '+sumLetArr);  const [sum1, sum2, ...remain] = sumLetArr;  console.log('3. sum1 : '+sum1+', sum2 : '+sum2+ ', remain : '+remain);  var varObj1 = { key1 : 'val1', key2 : 'val2' };  var varObj2 = { key2 : 'new2', key3 : 'val3' };  //javascript Object  var sumVarObj = Object.assign({}, varObj1, varObj2);  console.log('4. sumVarObj : '+JSON.stringify(sumVarObj));  //ES6 Object  var sumLetObj = {...varObj1, ...varObj2};  console.log('5. sumLetObj : '+JSON.stringify(sumLetObj));  var {key1, key3, ...others} = sumLetObj;  console.log('6. key1 : '+key1+', key3 : '+key3+', others : '+JSON.stringify(others));  }  render() {  return (  <h2>[THIS IS SpreadOperator ]</h2>  )  }  }  export default R011\_SpreadOperator; |

14~15: 기존 ES5에서 배열 2개를 합치기 위해서는 배열 각각에 인덱스로 접근에 값을 가져오거나 concat 함수를 이용한다. varArray1, varArray2 배열에 각각 인덱스(0, 1)로 접근해 인자 값(num1, num2, num3, num4)을 가져와 새로운 배열 sumVarArr에 하나씩 넣는다.

18: ES6에서는 전개 연산자 …(마침표 3개)을 배열명 앞에 붙여 여러 개의 배열을 합칠 수 있다.

20: sumLetArr 배열의 값을 추출에 개별 변수에 넣는다. 순서대로 변수 sum1에 sumLetArr[0] 값을 sum2에 sumLetArr[1] 값을 대입한다. 나머지 배열 값은 마지막에 전개 연산자 처리된 …remain 변수에 넣는다.

26: 기존 ES5에서 객체 2개를 합치기 위해서는 Object.assign() 함수를 이용해야 한다. 첫 번째 인자 {}는 함수의 return 값이고 뒤에 인자에 객체들을 ,(콤마)로 연결해 나열하면 여러 개의 객체를 합칠 수 있다.

29: ES6에서는 …(마침표 3개)을 객체명 앞에 붙여 여러 개의 객체를 합칠 수 있다.

31: sumLetObj 객체의 키와 값을 추출해 키와 동일한 명칭의 개별 변수에 넣는다. 나머지는 마지막에 전개 연산자 처리된 …others 변수에 넣는다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 22>**

012. class 사용하기

**• 학습 내용:** 코드를 객체 단위로 재사용하는 class에 대해 이해한다.

**• 힌트 내용:** 기존 ES5의 prototype과 비교해 ES6 class의 장점과 사용법을 확인한다.

기존 ES5 자바스크립트에서는 객체를 구현하기 위해 prototype을 사용한다. 객체는 상속을 통해 코드를 재사용할 수 있게 해준다. ES6에서 등장한 class는 prototype과 같은 개념인데, 쉽게 읽고 표현하기 위해 고안된 문법이다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ClassPrototype from './R012\_Class&Prototype'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ClassPrototype/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R012\_Class&Prototype.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R012\_Class&Prototype.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R012\_Class&Prototype.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | import React, { Component } from 'react';  class ClassPrototype extends Component {  constructor(props) {  super(props);  this.state = {};  }  componentDidMount() {  //ES5 prototype  var ExamCountFunc = (function () {  function ExamCount(num) {  this.number = num;  }  ExamCount.prototype.showNum = function () {  console.log('1. react\_' + this.number);  };  return ExamCount;  }());    var cnt = new ExamCountFunc('200');  cnt.showNum();  //ES6 class  class ExamCountClass {  constructor(num2) {  this.number2 = num2;  }  showNum() {  console.log(`2. react\_${this.number2}`);  }  }    var cnt2 = new ExamCountClass('2hundred');  cnt2.showNum();  }  render() {  return (  <h2>[THIS IS Class ]</h2>  )  }  }  export default ClassPrototype; |

|  |
| --- |
| NOTE |
| 자바스크립트에서는 함수를 객체로 사용할 수 있다. |

22: ExamCountFunc() 함수( 객체)를 실행한 후 return되는 결괏값( 객체)을 cnt라는 변수에 저장한다

13~15: 생성자 함수를 실행하는데, 파라미터로 전달받은 num 변수의 값(200)을 객체 변수 number에 저장한다.

23: 객체 안에 선언된 showNum() 함수를 실행한다.

16: 생성자 함수명.prototype. 함수명 형태로 선언해주면, 객체 외부에서 함수(cnt.showNum();)를 실행해 객체 내부에 선언된 함수를 사용할 수 있다. 함수가 실행되면, 생성자 함수에서 ‘200’으로 할당된 객체 변수 number를 사용한다.

35: ExamCountClass 객체를 생성한 후 객체를 cnt2라는 변수에 저장한다.

26: ES6에서는 class로 객체를 선언한다.

27~29: constructor()라는 생성자 함수가 실행되고 파라미터로 전달받은 num2이라는 변숫값(2hundred)을 객체 변수 number2에 저장한다.

36: 객체 안에 선언된 showNum() 함수를 실행한다.

30~32: 객체에 접근할 때 실행할 함수(showNum)는 class의 {} 괄호(scope) 안에 간단하게 선언할 수 있다. 함수가 실행되면, 생성자 함수에서 ‘2hundred’으로 할당된 객체 변수 number2를 사용한다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 23>**

013. 화살표 함수 사용

**• 학습 내용:** ES6에 추가된 화살표 함수의 사용 방법을 이해한다.

**• 힌트 내용:** 기존 ES5의 기본 function과 비교해 ES6 화살표 함수의 장점과 사용법을 확인한다.

ES6에서 등장한 화살표 함수는 ‘function’ 대신 ‘=>’ 문자열을 사용하며 ‘return’ 문자열을 생략할 수도 있다. 따라서 기존 ES5 함수보다 간략하게 선언할 수 있다. 또 화살표 함수에서는 콜백 함수에서 this를 bind해야 하는 문제도 발생하지 않는다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ArrowFunc from './R013\_ArrowFunction'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ArrowFunc/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R012\_Class&Prototype.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R013\_ArrowFunction.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R013\_ArrowFunction.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | import React, { Component } from 'react';  class R013\_ArrowFunction extends Component {  constructor(props) {  super(props);  this.state = {  arrowFuc: 'React200',  num: 3  };  }  componentDidMount() {  Function1(1);  this.Function2(1,1);  this.Function3(1,3);  this.Function4();  this.Function5(0,2,3);  function Function1(num1) {  return console.log(num1+'. Es5 Function');  }  }    Function2 = (num1, num2) => {  let num3 = num1 + num2;  console.log(num3+'. Arrow Function : '+this.state.arrowFuc);  }    Function3() {  var this\_bind = this;  setTimeout(function() {  console.log(this\_bind.state.num+'. Es5 Callback Function noBind : ');  console.log(this.state.arrowFuc);  },100);  }    Function4() {  setTimeout(function() {  console.log('4. Es5 Callback Function Bind : '+this.state.arrowFuc);  }.bind(this),100);  }    Function5 = (num1, num2, num3) => {  const num4 = num1 + num2 + num3;  setTimeout(() => {  console.log(num4+'. Arrow Callback Function : '+this.state.arrowFuc);  }, 100);  }  render() {  return (  <h2>[THIS IS ArrowFunction ]</h2>  )  }  }  export default R013\_ArrowFunction; |

14~18: Function 1~5까지의 함수를 순서대로 호출한다.

20~22: 함수를 호출할 때 전달받은 num1이라는 변수를 함수 내부에서 사용할 수 있다.

25~28: 함수를 ‘function’ 문자열 대신 ‘=>’로 선언했다. 함수 내에서 사용하는 this는 R013\_ArrowFunction 컴포넌트인데, this로 컴포넌트의 state 변수에 접근해 사용할 수 있다.

31~33: 콜백 함수 내부에서는 this로 컴포넌트에 접근할 수 없기 때문에 접근할 수 있는 변수에 this를 백업한다. 백업된 변수 this\_bind를 이용해 컴포넌트의 state 변수에 접근할 수 있다.

34: 콜백 함수 내부에서 this는 window 객체이기 때문에 this로 state 변수에 접근하면 undefined에러가 발생한다.

40~41: 콜백 함수에 함수 밖의 this를 bind해주면, this를 컴포넌트로 사용할 수 있다.

47: 화살표 함수에서는 this를 bind해주지 않아도 this를 컴포넌트로 사용해 state 변수에 접근할 수 있다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 24>**

014. forEach() 함수 사용하기

**• 학습 내용:** 배열 함수인 forEach() 함수의 사용 방법을 이해한다.

**• 힌트 내용:** for문과 비교해 forEach() 함수의 장점과 사용법을 확인한다.

배열 함수 forEach()는 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않는다. 배열의 처음부터 마지막 순번까지 모두 작업하는 경우 forEach()문을 사용하는 것이 간편하다. 하지만 특정 순번에서만 배열 값을 사용하거나 변경해야 하는 상황이라면 for문을 사용해야 한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ForEach from './R014\_ForEach'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ForEach/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R014\_ForEach.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R014\_ForEach.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R014\_ForEach.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | import React, { Component } from 'react';  class R014\_ForEach extends Component {  componentDidMount() {  var For\_Arr = [3, 2, 8, 8 ];  var For\_newArr = [];    for (var i = 0; i < For\_Arr.length; i++) {  For\_newArr.push(For\_Arr[i]);  }  console.log("1. For\_newArr : ["+For\_newArr+"]");  var ForEach\_Arr = [3, 3, 9, 8 ];  var ForEach\_newArr = [];  ForEach\_Arr.forEach((result) => {  ForEach\_newArr.push(result);  })  console.log("2. ForEach\_newArr : ["+ForEach\_newArr+"]");  }  render() {  return (  <h2>[THIS IS ForEach ]</h2>  )  }  }  export default R014\_ForEach; |

9~11: for문에서는 순번 변수(i)와 배열의 크기(For\_Arr.length)가 필요하다. 순번을 0부터 1씩 증가시킨다. 배열의 크기보다 1이 작은 값이 될 때까지 새로운 함수(For\_newArr)에 기존 함수 값을 넣는다(push).

16~18: forEach 함수에서는 순번과 배열의 크기 정보를 사용하지 않는다. 0부터 배열의 크기만큼 반복하며 순서대로 배열 값을 반환한다. 반복문이 실행될 때마다 콜백 함수로 결괏값(result)을 받아 새로운 함수(ForEach\_newArr)에 넣는다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 25>**

015. map() 함수 사용하기

**• 학습 내용:** 배열 함수인 map() 함수의 사용 방법을 이해한다.

**• 힌트 내용:** map() 함수의 특징과 사용법을 확인한다.

배열 함수 map()은 forEach()와 마찬가지로 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않는다. 차이점은 map()은 forEach()와 달리 return을 사용해 반환 값을 받을 수 있다는 것이다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Map from './R015\_Map'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Map/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R015\_Map.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R015\_Map.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R015\_Map.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  class R015\_Map extends Component {  componentDidMount() {  var Map\_Arr = [3, 2, 8, 8 ]  let Map\_newArr = Map\_Arr.map(x => x)  console.log("1. Map\_newArr : ["+Map\_newArr+"]")  let Map\_mulitiArr = Map\_Arr.map(x => x \* 2)  console.log("2. Map\_mulitiArr : ["+Map\_mulitiArr+"]")  var ObjArray = [{key:'react', value:'200'},  {key:'리액트', value:'TwoHundred'}];  let Map\_objArr = ObjArray.map((obj, index) => {  console.log((index+3)+". obj : "+JSON.stringify(obj))  var Obj = {};  Obj[obj.key] = obj.value;  return Obj;  });  console.log("5. Map\_objArr : "+JSON.stringify(Map\_objArr))  }  render() {  return (  <h2>[THIS IS Map ]</h2>  )  }  }  export default R015\_Map; |

7: 기존 배열(Map\_Arr)에서 map() 함수를 사용해 순서대로 하나씩 요소에 접근해 가져온다. 이때마다 콜백 함수가 실행된다. 가져온 값을 변수 x에 넣은 후 그대로 반환해(x => x) 순서대로 쌓아 놓는다. 마지막 요소까지 반복했다면, 한 번에 새로운 배열(Map\_newArr)에 저장한다.

|  |
| --- |
| NOTE |
| 화살표 함수는 return 구문을 생략할 수 있다. Line 7에서 사용한 Map\_Arr.map(x => x)는 Map\_Arr.map(x => {return x})와 동일하게 동작한다. |

10: 기존 배열(Map\_Arr)의 요소에 순서대로 접근한 후 각각 2를 곱해 새로운 배열(Map\_mulitiArr)에 저장한다.

13~14: 배열 안에 객체를 생성한다.

15: 배열 안의 객체를 순서대로 가져와 콜백 함수를 실행하는데, 가져온 값을 obj라는 변수에 저장한다. 콜백 함수의 두 번째 인자인 index는 기존 배열의 인덱스와 동일하다.

16: 기존 배열에서 가져온 객체 값을 순서대로 출력한다.

17~19: 새로운 객체(Obj)를 선언한다. 기존 객체(ObjArray)의 key, value 값을 새로운 객체(Obj)의 key, value로 값으로 저장한다. 모든 반복이 끝나면 Map\_objArr 변수에 반환 값들을 저장한다.

크롬 웹 브라우저에서 [F12]를 눌러 개발자 도구 창을 띄운 후 [Console] 탭을 클릭하면 출력된 로그를 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 26>**

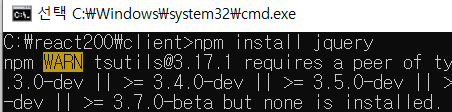
016. jquery 사용하기

**• 학습 내용:** react에서 jquery 사용 방법을 이해한다.

**• 힌트 내용:** jquery를 임포트하는 방법과 기본 문법을 확인한다.

jquery는 가장 인기 있는 자바스크립트 라이브러리다. 이벤트 처리, 애니메이션 등 자바스크립트의 기능을 간단하고 빠르게 구현할 수 있도록 지원해준다.

jquery를 사용하기 위해 층 창을 열어 client 폴더 경로로 이동한다. [npm install jquery]를 입력하면 다음과 같이 npm이 jquery를 설치한다.



**<그림 27>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Jquery from './R016\_Jquery'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Jquery/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R016\_Jquery.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R016\_Jquery.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R016\_Jquery.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import $ from 'jquery';  class R016\_Jquery extends Component {  input\_alert = (e) => {  var input\_val = $('#inputId').val();  alert(input\_val);  }  render() {  return (  <div>  <h2>[THIS IS Jquery ]</h2>  <input id="inputId" name="inputName"/>  <button id="buttonId" onClick={e => this.input\_alert(e)}>Jquery Button</button>  </div>  )  }  }  export default R016\_Jquery; |

2: jquery를 임포트해 $ 기호를 선언한다. jquery를 사용할 때 $ 기호를 붙여 사용해야 한다는 것을 의미한다.

15: 화면의 왼쪽에 보이는 버튼으로, 값을 입력하면 <input> 태그의 value 값으로 저장된다.

16: button을 클릭하면 input\_alert이라는 함수를 호출한다.

7~8: 함수가 호출되면 jquery 문법 $('#inputId')으로, id 값이 inputId인 <input> 태그에 접근한다. <input> 태그의 value 값(line 15에서 입력한 값)를 가져와 input\_val 변수에 저장한다. 변숫값을 팝업 창으로 표시한다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 28>**

초급 React.js 기초 다지기

(도비라)2부 초급: React.js 기초 다지기

**<저자 요청: 5~6줄 정도의 발문 부탁드립니다.>**

017. props 사용하기

**• 학습 내용:** props 사용 방법을 이해한다.

**• 힌트 내용:** props에 데이터를 넣는 부분과 받아오는 부분을 확인한다.

props는 부모 컴포넌트가 자식 컴포넌트에 데이터를 전달할 때 사용한다. props를 전달받은 자식 컴포넌트에서는 데이터를 수정할 수 없다. 데이터를 변경하기 위해서는 컴포넌트 내부에서만 사용하는 변수에 값을 넣어 사용해야 한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Props from './R017\_Props'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Props props\_val="THIS IS PROPS"/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R017\_Props.js 파일을 임포트해 사용할 수 있도록 한다.

10: line 3에서 임포트한 하위 컴포넌트 R017\_Props에 전달할 props 변수(props\_val)에 값(THIS IS PROPS)을 저장한다.

src 폴더에 R017\_Props.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R017\_Props.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13 | import React, { Component } from 'react';  class R017\_Props extends Component {  render() {  let props\_value = this.props.props\_val;  props\_value += ' from App.js'  return (  <div>{props\_value}</div>  )  }  }  export default R017\_Props; |

5: this.props. 뒤에 상위 컴포넌트(App.js)에서 전달받은 props 변수명을 붙이면, 해당 데이터를 사용할 수 있다.

6: 데이터를 수정해야 할 경우, props 자체가 아닌 컴포넌트 내부 변수(props\_value)에 옮겨 가공한다. App.js에서 넘겨받은 문자열(THIS IS PROPS) 뒤에 새로운 문자열( from App.js)를 붙인다.

8: line 6에서 가공된 문자열을 화면에 표시시킨다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 29>**

018. props 자료형 선언하기

**• 학습 내용:** props자료형 선언 방법을 이해한다.

**• 힌트 내용:** 선언한 자료형과 다른 자료형이 전달됐을 때 경고 메시지를 확인한다.

자식 컴포넌트에서 props에 대한 자료형을 선언해 놓으면, 부모 컴포넌트에서 넘어오는 props 변수들의 자료형과 비교한다. 이때 자료형이 일치하지 않는다면, 경고 메시지로 알려주기 때문에 잘못된 데이터를 확인할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React from 'react';  import './App.css';  import PropsDatatype from './R018\_PropsDatatype'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsDatatype  String="react"  Number={200}  Boolean={1==1}  Array={[0, 1, 8]}  Object Json={{react:"리액트", twohundred:"200"}}  Function={console.log("FunctionProps: function!")}  />  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R018\_PropsDatatype.js 파일을 임포트해 사용할 수 있도록 한다.

10~17: 문자열, 숫자, 불리언, 배열, 객체, 함수 데이터를 props에 담아 하위 컴포넌트로 전달한다.

src 폴더에 R018\_PropsDatatype.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R018\_PropsDatatype.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  import datatype from 'prop-types';  class R018\_PropsDatatype extends Component {  render() {  let {  String, Number, Boolean, Array, Object Json, Function  } = this.props  return (  <div style={{padding: "0px"}}>  <p>StringProps: {String}</p>  <p>NumberProps: {Number}</p>  <span>BooleanProps: {Boolean.toString()}</span>  <p>ArrayProps: {Array.toString()}</p>  <p>Object JsonProps: {JSON.stringify(Object Json)}</p>  <p>FunctionProps: {Function}</p>  </div>  )  }  }  R018\_PropsDatatype.propTypes = {  String: datatype.number,  Number: datatype.number,  Boolean: datatype.bool,  Array: datatype.array,  Object Json: datatype.object,  Function: datatype.func,  }  export default R018\_PropsDatatype; |

6~8: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값들을 할당한다.

22~29: props 값들 각각에 적합한 자료형을 선언한다. Line 23에서 문자열 타입인 props.String 변수의 자료형을 number로 선언해 경고 메시지가 발생할 수 있도록 한다. 경고 메시지는 개발자 도구 콘솔 창에서 확인할 수 있다.

|  |
| --- |
| NOTE |
| props의 자료형 선언을 위해 line 2에서 리액트 기본 내장 패키지인 prop-types를 임포트해 사용했다. proTypes가 데이터 유효성 검증을 하고 콘솔에 경고 메시지를 출력한다. |

11~16: 부모 컴포넌트에서 전달한 props 변수들을 화면에 표시시킨다. Line 11의 String 변수는 line 23에서 자료형이 일치하지 않았다. 하지만 에러가 아닌 경고 메시지이기 때문에 화면에 String 변수가 정상적으로 표시된다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 30>**

019. props Boolean으로 사용하기

**• 학습 내용:** props Boolean 사용 방법을 이해한다.

**• 힌트 내용:** props 다른 자료형들이 갖고 있지 않은 boolean 변수만의 특징을 확인한다.

props 값을 boolean형으로 하위 컴포넌트로 전달할 경우, true나 false 중 하나를 할당한다. 추가 문법으로 props 변수를 선언한 후 값을 할당하지 않고 넘기면 true가 기본값으로 할당된다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import PropsBoolean from './R019\_PropsBoolean'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsBoolean BooleanTrueFalse={false}/>  <PropsBoolean BooleanTrueFalse/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R019\_PropsBoolean.js 파일을 임포트해 사용할 수 있도록 한다.

10: BooleanTrueFalse 변수를 선언하고 false 값을 넣은 후 props에 담아 하위 컴포넌트로 전달한다.

11: BooleanTrueFalse 변수에 선언하고 할당 없이 props에 담아 하위 컴포넌트로 전달한다.

src 폴더에 R019\_PropsBoolean.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R019\_PropsBoolean.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  class R019\_PropsDatatype extends Component {  render() {  let {  BooleanTrueFalse  } = this.props  return (  <div style={{padding: "0px"}}>  {BooleanTrueFalse ? '2. ' : '1. '}  {BooleanTrueFalse.toString()}  </div>  )  }  }  export default R019\_PropsDatatype; |

5~7: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값을 할당한다.

10: 삼항 연산자를 이용해 BooleanTrueFalse 변수가 true이면 ‘2‘, false면 ‘1‘을 화면에 출력한다.

|  |
| --- |
| NOTE |
| 삼항 연산자란 [조건문? 조건문이 참일 때 값 : 조건문이 거짓일 때 값] 형태로 사용한다. if문을 사용하는 것보다 코드가 간결해지지만, 조건문이 길어질 경우 가독성이 떨어질 수 있다. |

11: BooleanTrueFalse 변수가 false일 경우 그대로 false가 출력되고, 값이 없을 경우 기본값으로 true가 화면에 출력된다. boolean 변수는 직접 화면에 출력할 수 없으므로 출력을 하기 위해 toString() 함수를 사용해 문자열로 변환한다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 31>**

020. props 객체형으로 사용하기

**• 학습 내용:** props 객체형 사용 방법을 이해한다.

**• 힌트 내용:** props 객체 자체의 자료형이 아닌, 객체 내부 변수들의 자료형 선언 방법을 확인한다.

props 값을 객체로 하위 컴포넌트로 전달할 경우, 자료형을 object로 선언한다. 하지만 객체의 형태( 객체 내부 변수들)의 자료형을 선언할 때는 shape이라는 타입을 사용한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import PropsObjVal from './R020\_PropsObjVal'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsObjVal Object Json={{react:"리액트", twohundred:"200"}}/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R020\_PropsObjVal.js 파일을 임포트해 사용할 수 있도록 한다.

10: Object Json 변수와 key를 선언하고 value를 할당한 후 props에 담아 하위 컴포넌트로 전달한다.

src 폴더에 R020\_PropsObjVal.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R020\_PropsObjVal.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import React, { Component } from 'react';  import datatype from 'prop-types';  class R020\_PropsObjVal extends Component {  render() {  let {  Object Json  } = this.props  return (  <div style={{padding: "0px"}}>  {JSON.stringify(Object Json)}  </div>  )  }  }  R020\_PropsObjVal.propTypes = {  Object Json: datatype.shape({  react: datatype.string,  twohundred: datatype.number  })  }  export default R020\_PropsObjVal; |

6~8: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값을 할당한다.

18~21: shape 타입을 사용해 객체 변수 Object Json의 내부 key 값(react, twohundred)에 대해 자료형을 선언한다. twohundred가 문자열(“200”)로 전달됐지만, 자료형이 number로 선언됐다. 자료형이 일치하지 않아 경고 메시지가 발생한다. 경고 메시지는 개발자 도구 콘솔 창에서 확인할 수 있다.

11: Object Json 객체의 key와 value 값들을 화면에 출력한다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 32>**

021. props 필수 값으로 사용하기

**• 학습 내용:** props를 필수 값으로 사용하는 방법을 이해한다.

**• 힌트 내용:** props를 필수 값으로 지정하는 문법을 확인한다.

props의 자료형을 선언할 때 prop-types를 사용한다. 자료형 설정 대신 isRequired를 조건으로 추가하면, 변숫값이 없는 경우 경고 메시지가 발생할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import PropsRequired from './R021\_PropsRequired'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsRequired ReactNumber={200}/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R021\_PropsRequired.js 파일을 임포트해 사용할 수 있도록 한다.

10: ReactNumber 변수에 숫자 200을 할당한 후 props에 담아 하위 컴포넌트로 전달한다.

src 폴더에 R021\_PropsRequired.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R021\_PropsRequired.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import datatype from 'prop-types';  class R021\_PropsRequired extends Component {  render() {  let {  ReactString,  ReactNumber  } = this.props  return (  <div style={{padding: "0px"}}>  {ReactString}{ReactNumber}  </div>  )  }  }  R021\_PropsRequired.propTypes = {  ReactString: datatype.isRequired,  }  export default R021\_PropsRequired; |

6~8: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값을 할당한다.

18~20: ReactString이라는 props 값을 필수 값으로 지정한다. 하지만 상위 컴포넌트에서 ReactSring이라는 변수를 전달하지 않았기 때문에 경고 메시지가 발생한다. 경고 메시지는 개발자 도구 콘솔 창에서 확인할 수 있다.

12: ReactSring 변숫값은 비어 있기 때문에 상위 컴포넌트에서 전달받은 ReactNumber 변숫값(200)만 화면에 출력된다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 33>**

022. props 기본값 정의하기

**• 학습 내용:** props를 기본값으로 사용하는 방법을 이해한다.

**• 힌트 내용:** props를 기본값으로 지정하는 문법을 확인한다.

props의 기본값은 부모 컴포넌트에서 값이 넘어 오지 않았을 때 사용한다. defaultProps라는 문법을 사용한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import PropsDefault from './R022\_PropsDefault'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsDefault ReactNumber={200}/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R022\_PropsDefault.js 파일을 임포트해 사용할 수 있도록 한다.

10: ReactNumber 변수에 숫자 200을 할당한 후 props에 담아 하위 컴포넌트로 전달한다.

src 폴더에 R021\_PropsRequired.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R022\_PropsDefault.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  class R022\_PropsDefault extends Component {  render() {  let {  ReactString,  ReactNumber  } = this.props  return (  <div style={{padding: "0px"}}>  {ReactString}{ReactNumber}  </div>  )  }  }  R022\_PropsDefault.defaultProps = {  ReactString: "리액트",  ReactNumber: 400  }  export default R022\_PropsDefault; |

6~8: render() 함수 내에서 지역 변수를 선언해 props로 전달된 값을 할당한다.

17~20: 상위 컴포넌트에서 값이 전달될 것이라 기대되는 ReactString과 ReactNumber 변수에 각각 기본값을 할당했다.

11: ReactSring 변숫값은 비어 있기 때문에 line 18에서 지정한 기본값으로 화면에 표시된다. ReactNumber 변숫값은 상위 컴포넌트에서 전달됐기 때문에 line 19에서 지정한 기본값은 무시된다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 34>**

023. props 자식 Component에 node 전달하기

**• 학습 내용:** props에 node를 사용하는 방법을 이해한다.

**• 힌트 내용:** props에 node를 넣어 전달하는 문법을 확인한다.

props를 하위 컴포넌트 태그 안쪽에 선언해 전달하는 것 이외에도 하위 컴포넌트 태그 사이에 작성된 node를 전달할 수 있다.

|  |
| --- |
| NOTE |
| node란, html 문서를 구성하는 포괄적인 개념이다. 노드의 종류는 문서 요소, 속성, 텍스트, 주석 등이 있다. 예를 들면, <p id=”a”>bc</p>에서 <p> 태그가 요소, id가 속성, bc가 텍스트다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React from 'react';  import './App.css';  import PropsNode from './R023\_PropsNode'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PropsNode>  <span>node from App.js</span>  </PropsNode>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R023\_PropsNode.js 파일을 임포트해 사용할 수 있도록 한다.

10~12: 하위 컴포넌트 태그 사이에 <span> 태그를 추가하면 props에 담아 하위 컴포넌트에 전달한다.

src 폴더에 R023\_PropsNode.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R023\_PropsNode.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13 | import React, { Component } from 'react';  class R023\_PropsNode extends Component {  render() {  return (  <div style={{padding: "0px"}}>  {this.props.children}  </div>  )  }  }  export default R023\_PropsNode; |

7: 상위 컴포넌트에서 전달한 노드는 this.props.children이라는 문법으로 접근해 사용할 수 있다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 35>** |

024. state 사용하기

**• 학습 내용:** state 사용 방법을 이해한다.

**• 힌트 내용:** state 값을 화면에 표시시키는 문법을 확인한다.

Props를 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달할 때 사용했다면, state는 하나의 컴포넌트 안에서 전역 변수처럼 사용한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ReactState from './R024\_ReactState'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactState reactString={"react"}/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R024\_ReactState.js 파일을 임포트해 사용할 수 있도록 한다.

10: reactString이라는 변수를 선언하고 react라는 문자열을 할당한 후 props에 담아 하위 컴포넌트에 전달한다.

src 폴더에 R024\_ReactState.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R024\_ReactState.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React, { Component } from 'react';  class R024\_ReactState extends Component {  constructor (props) {  super(props);  this.state = {  StateString: this.props.reactString,  StateNumber: 200,  }  }  render() {  return (  <div style={{padding: "0px"}}>  {this.state.StateString}{this.state.StateNumber}  </div>  )  }  }  export default R024\_ReactState; |

6~9: 가장 먼저 실행되는 생성자 함수 constructor 안에서 state 변수의 초깃값을 정의해야 한다. StateString 변수에는 props로 전달된 reactString 값을 저장하고 StateNumber 변수에는 숫자 200을 저장한다.

15: this.state.변수명 문법으로 state 변수에 접근한다. state 값을 화면에 그대로 표시시킨다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 36>**

025. setState() 함수 사용하기

**• 학습 내용:** setState() 함수를 사용하는 방법을 이해한다.

**• 힌트 내용:** state를 직접 변경했을 때와 setState() 함수를 사용했을 때의 차이를 확인한다.

this.state.변수명 = value와 같이 직접 state를 변경하면 render() 함수를 호출하지 않으므로 화면에 보이는 state 값은 바뀌기 전 상태로 남게 된다. setState() 함수로 state를 변경해야 render() 함수를 호출해 변경된 값을 화면에 보여줄 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import SetState from './R025\_SetState'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <SetState/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R025\_SetState.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R025\_SetState.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R025\_SetState.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import React, { Component } from 'react';  class R025\_SetState extends Component {  constructor (props) {  super(props);  this.state = {  StateString: 'react',  }  }  StateChange = (flag) => {  if(flag == 'direct') this.state.StateString = '리액트';  if(flag == 'setstate') this.setState({StateString : '리액트'});  }  render() {  return (  <div style={{padding: "0px"}}>  <button onClick={(e) => this.StateChange('direct', e)}>state 직접 변경</button>  <button onClick={(e) => this.StateChange('setstate', e)}>setState로 변경</button><br/>  [state 변경하기] StateString : {this.state.StateString}  </div>  )  }  }  export default R025\_SetState; |

21: line 7에서 state 변수 StateString의 초깃값으로 react라는 문자열을 저장했다. constructor() 함수가 실행되고 render() 함수에서 화면을 그리기 때문에 {this.state.StateString}은 react로 표시된다.

12: line 19에서 [state 직접 변경] 버튼을 눌렀을 때 this.state.변수명 = value 문법으로 state를 직접 변경한다. 이때 StateString 값은 ‘리액트’로 변경되지만 render() 함수를 호출하지 않으므로 화면에는 이전 값인 ‘react’로 표시된다.

13: line 20에서 [setState로 변경] 버튼을 눌렀을 때 setState() 함수로 state를 변경한다. 이때 StateString 값은 ‘리액트’로 변경되고 render() 함수를 다시 호출해 화면에는 변경된 값인 ‘리액트’로 표시된다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 37>**  **<그림 38>** |

026. state를 직접 변경한 후 forceUpdate() 함수 사용하기

**• 학습 내용:** forceUpdate() 함수 사용 방법을 이해한다.

**• 힌트 내용:** state를 직접 변경했을 때 forceUpdate()로 render() 함수를 실행한다.

this.state.변수명 = value와 같이 직접 state를 변경하면 render() 함수를 호출하지 않으므로 화면에 보이는 state 값은 바뀌기 전 상태로 남게 된다. 이때 forceUpdate() 함수로 화면을 새로 고침하면, render() 함수를 호출해 변경된 값을 화면에 보여줄 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ForceUpdate from './R026\_ForceUpdate'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ForceUpdate/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R026\_ForceUpdate.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R026\_ForceUpdate.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R026\_ForceUpdate.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import React, { Component } from 'react';  class R026\_ForceUpdate extends Component {  constructor (props) {  super(props);  this.state = {  StateString: 'react',  }  }  StateChange = () => {  this.state.StateString = '리액트';  this.forceUpdate();  }  render() {  return (  <div style={{padding: "0px"}}>  <button onClick={(e) => this.StateChange('direct', e)}>state 직접 변경</button><br/>  [state 변경하기] StateString : {this.state.StateString}  </div>  )  }  }  export default R026\_ForceUpdate; |

21: line 7에서 state 변수 StateString의 초깃값으로 react라는 문자열을 저장했다. constructor() 함수가 실행되고 render() 함수에서 화면을 그리기 때문에 {this.state.StateString}은 react로 표시된다.

12: line 19에서 [state 직접 변경] 버튼을 눌렀을 때 this.state.변수명 = value 문법으로 state를 직접 변경한다. 이때 StateString 값은 ‘리액트’로 변경된다.

13: line 12까지만 실행하면 state 값은 변경되지만, render() 함수가 실행되지 않으므로 line 20 화면에 표시되는 state 값은 변하지 않는다. forceUpdate() 함수는 화면을 강제로 새로 고침하기 때문에 render() 함수를 다시 실행시켜 화면에 변경된 state 값을 표시시킬 수 있다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 39>**  **<그림 40>** |

027. Compoent(class형 컴포넌트)

**• 학습 내용:** Class형 컴포넌트 중 Component를 사용하는 방법을 이해한다.

**• 힌트 내용:** Component와 render() 함수와의 관계를 확인한다.

class형 컴포넌트에는 Component와 PureComponent가 있다. 두 컴포넌트 모두 props와 state의 변경에 따라 render() 함수를 호출하는데, 변경에 대한 기준이 다르다. Component에서는 비교 대상이 완전히 동일하지 않으면 변경이 발생했다고 본다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ComponentClass from './R027\_ComponentClass'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ComponentClass/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R027\_ComponentClass.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R027\_ComponentClass.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R027\_ComponentClass.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  class R027\_ComponentClass extends Component {  constructor (props) {  super(props);  this.state = {  StateString: 'react',  StateArrayObj: ['react', { react: '200' }]  }  }  buttonClick = (type) => {  if(type === 'String'){  this.setState({ StateString: 'react' });  }else{  this.setState({ StateArrayObj: ['react', { react: '200' }] });  }  }  render() {  console.log('render() 실행')  return (  <div>  <button onClick={e => this.buttonClick('String')}> 문자열 변경</button>  <button onClick={e => this.buttonClick('ArrayObject')}> 객체 배열 변경</button>  </div>  )  }  }  export default R027\_ComponentClass; |

1~3: Component 클래스를 임포트해 상속받아 사용한다.

7~8: state에 문자열 변수와 객체가 포함된 배열 변수를 선언하고 초깃값을 저장한다.

13~14: line 24에서 [문자열 변경] 버튼을 클릭하면, line 7에서 선언한 문자열 변수와 동일한 state 값을 setState() 함수로 선언 및 초기화한다. 이때 컴포넌트는 state 값이 변경됐다고 간주해 render() 함수를 실행한다.

15~16: line 25에서 [객체 배열 변경] 버튼을 클릭하면, line 8에서 선언한 배열 변수와 동일한 state 값을 setState() 함수로 선언 및 초기화한다. 이때 컴포넌트는 state 값이 변경됐다고 간주해 render() 함수를 실행한다.

|  |
| --- |
| NOTE |
| setState() 함수로 실행한 값은 이전에 있던 state 변숫값과 동일하지만 Component에서는 새로운 state 변수가 같은 이름으로 생성됐다고 인식한다. |

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다. 두 가지 버튼을 클릭하면 render() 함수가 실행돼 line 21에서 콘솔 로그를 출력한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 41>**

028. PureComponent(class형 컴포넌트)

**• 학습 내용:** Class형 컴포넌트 중 PureComponent를 사용하는 방법을 이해한다.

**• 힌트 내용:** PureComponent와 render() 함수와의 관계를 확인한다.

class형 컴포넌트에는 Component와 PureComponent가 있다. 두 컴포넌트 모두 props와 state의 변경에 따라 render() 함수를 호출하는데, 변경에 대한 기준이 다르다. PureComponent에서는 비교 대상의 값을 비교해 동일하지 않으면 변경이 발생했다고 본다. 불필요한 render() 함수 실행을 줄여 페이지 성능을 증가시킬 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import PureComponentClass from './R028\_PureComponentClass'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <PureComponentClass/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R028\_PureComponentClass.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R028\_PureComponentClass.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R028\_PureComponentClass.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { PureComponent } from 'react';  class R028\_PureComponentClass extends PureComponent {  constructor (props) {  super(props);  this.state = {  StateString: 'react',  StateArrayObj: ['react', { react: '200' }]  }  }  buttonClick = (type) => {  if(type === 'String'){  this.setState({ StateString: 'react' });  }else{  this.setState({ StateArrayObj: ['react', { react: '200' }] });  }  }  render() {  console.log('render() 실행')  return (  <div>  <button onClick={e => this.buttonClick('String')}> 문자열 변경</button>  <button onClick={e => this.buttonClick('ArrayObject')}> 객체 배열 변경</button>  </div>  )  }  }  export default R028\_PureComponentClass; |

1~3: PureComponent 클래스를 임포트해 상속받아 사용한다.

7~8: state에 문자열 변수와 객체가 포함된 배열 변수를 선언하고 초깃값을 저장한다.

13~14: line 24에서 [문자열 변경] 버튼을 클릭하면, line 7에서 선언한 문자열 변수와 동일한 state 값을 setState() 함수로 선언 및 초기화한다. 이때 컴포넌트는 state 값이 동일하다고 간주해 render() 함수를 실행하지 않는다. PureComponent에서는 새로운 state 변수가 선언됐지만, 변숫값만 비교한다.

15~16: line 25에서 [객체 배열 변경] 버튼을 클릭하면, line 8에서 선언한 배열 변수와 동일한 state 값을 setState() 함수로 선언 및 초기화한다. 이때 컴포넌트는 state 값이 변경됐다고 간주해 render() 함수를 실행한다. PureComponent에서는 기본적으로 변숫값만 비교하지만, 예외적으로 객체에 대해서는 참조 값을 비교한다. Line 16에서 생성한 객체({ react: '200' })의 데이터는 같지만, 참조 값이 달라 다른 객체로 인식된다.

|  |
| --- |
| NOTE |
| 참조 값이란, 객체를 생성했을 때 저장되는 메모리 주소다. 완전히 동일한 객체라도 새로 선언하면 다른 참조 값을 갖는다. |

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다. 두 가지 버튼을 클릭하면 render() 함수가 실행돼 line 21에서 콘솔 로그를 출력한다.

<실행 결과>

|  |
| --- |
| **빈 박스 넣어주세요 저자에게 그림을 확인해야 합니다.<그림41과 중복?>** |

029. shallow-equal 사용하기(class형 컴포넌트)

**• 학습 내용:** shallow-equal의 비교 함수를 사용하는 방법을 이해한다.

**• 힌트 내용:** shallow-equal의 비교 함수를 사용해 render() 함수 실행을 결정할 수 있다.

shallow-equal 패키지는 PureComponent에서 state 값의 변경을 비교하는 것과 동일한 기능을 하는 함수를 제공한다. shallowEqualArrays() 함수를 사용하면 문자열과 배열은 값만 비교한다. 객체는 PureComponent와 동일하게 참조 값을 비교한다.

|  |
| --- |
| INSTALL |
| 명령 프롬프트를 연 후 cd C:\react200\client 명령어를 이용해 설치 경로로 이동한다. npm install shallow-equal 명령어로 shallow-equal 패키지를 설치한다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ShallowEqual from './R029\_ShallowEqual'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ShallowEqual/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R029\_ShallowEqual.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R029\_ShallowEqual.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R029\_ShallowEqual.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  import { shallowEqualArrays} from "shallow-equal";  class R029\_ShallowEqual extends Component {  constructor (props) {  super(props);  this.state = { StateString: 'react' }  }  shouldComponentUpdate(nextProps, nextState){  return !shallowEqualArrays(this.state, nextState)  }  componentDidMount(){  const object = { react : '200'};  const Array1 = ['리액트', object];  const Array2 = ['리액트', object];  const Array3 = ['리액트', { react : '200'}];    console.log('shallowEqualArrays(Array1, Array2) : ' + shallowEqualArrays(Array1, Array2));  console.log('shallowEqualArrays(Array2, Array3) : ' + shallowEqualArrays(Array2, Array3));  this.setState({StateString : 'react'})  }  render() {  console.log('render() 실행')  return (<div></div>)  }  }  export default R029\_ShallowEqual; |

2: shallow-equal 임포트해 사용할 수 있도록 한다.

20: line 16, 17에서 선언한 배열의 값을 비교한다. 이때 object 변수는 동일한 객체를 참조한다. shallowEqualArrays() 함수로 두 배열을 비교하고 동일하면 true를 반환한다.

21: line 17, 18에서 선언한 배열의 값을 비교한다. 이때 line 17의 object와 line 18의 { react : '200'}는 값은 같지만, 서로 다른 참조 값을 갖는 다른 객체다. shallowEqualArrays() 함수 비교 결과 false를 반환한다.

10~12: shouldComponentUpdate() 함수는 변경된 props와 state 값을 파라미터로 받을 수 있고 반환 값이 true일 때 render() 함수를 실행시킨다. line 22에서 문자열 state 값이 변경됐는데, 원래 Component 클래스에서는 변경으로 인식해 render() 함수를 실행시킨다. 그런데 shouldComponentUpdate() 함수에서 shallowEqualArrays() 함수로 값만 비교하기 때문에 false를 반환하고 render() 함수를 실행시키지 않는다.

|  |
| --- |
| NOTE |
| Component 클래스에서도 shouldComponentUpdate(), shallowEqualArrays() 함수를 사용하면, PureComponent 클래스처럼 값만 비교해 render() 함수를 실행시킬 수 있다. |

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다. 처음 페이지 로딩 때 render() 함수가 실행되고 line 22에서 state 값이 변경됐을 때 실행되지 않는 것을 확인할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 42>** |

030. 함수형 컴포넌트 사용하기

**• 학습 내용:** 함수형 컴포넌트를 사용하는 방법을 이해한다.

**• 힌트 내용:** 클래스형 컴포넌트와 비교해 함수형 컴포넌트의 특징을 확인한다.

함수형 컴포넌트는 클래스형 컴포넌트와 달리, state가 없고 생명주기 함수를 사용할 수 없다. 상위 컴포넌트에서 props와 context를 파라미터로 전달받아 사용하고 render() 함수가 없어 return만 사용해 화면을 그려준다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import FunctionComponent from './R030\_FunctionComponent'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <FunctionComponent contents="[THIS IS FunctionComponent ]"/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R030\_FunctionComponent.js 파일을 임포트해 사용할 수 있도록 한다.

10: R030\_FunctionComponent 컴포넌트에 contents 변수를 props로 전달한다.

src 폴더에 R030\_FunctionComponent.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R030\_FunctionComponent.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10 | import React from 'react';  function R030\_FunctionComponent(props){  let { contents } = props;  return (  <h2>{contents}</h2>  )  }  export default R030\_FunctionComponent; |

3~7: 함수형 컴포넌트는 return 값이 있는 function과 동일한 구조를 갖는다.

4: 상위 컴포넌트에서 전달받은 props를 지역 변수에 할당한다. 클래스형 컴포넌트와 달리, props 앞에 this가 붙지 않는다.

6: line 4에서 할당받은 변수 contents 값을 화면에 표시한다.

<실행 결과>

|  |
| --- |
| **<그림 43>** |

031. Hook 사용하기

**• 학습 내용:** hook을 사용하는 방법을 이해한다.

**• 힌트 내용:** 함수형 컴포넌트에서 사용되는 hook 기능을 확인한다.

함수형 컴포넌트에서 클래스형 컴포넌트와 같이 state와 생명주기 함수와 같은 기능을 사용하기 위해 hook을 이용한다. 대표적인 hook 함수로는 useState()와 useEffect()가 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import ReactHook from './R031\_ReactHook'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactHook/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R031\_ReactHook.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R031\_ReactHook.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R031\_ReactHook.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18 | import React, { useState, useEffect } from 'react';  function R031\_ReactHook(props){  const [contents, setContents] = useState('[THIS IS REACT ]');  useEffect(() => {  console.log('useEffect');  });  return (  <div style={{padding: "0px"}}>  <h2>{contents}</h2>  <button onClick={() =>setContents('[THIS IS HOOK ]')}> 버튼</button>  </div>  )  }  export default R031\_ReactHook; |

1: react에서 기본으로 제공해주는 useState()와 useEffect() 함수를 사용하기 위해 임포트해준다.

4: useState() 함수로 state 변숫값을 선언 및 할당한다. 이때 두 가지 인자가 선언된 배열이 반환된다. 첫 번째 인자 contents는 state 변수명이고 두 번째 인자 setContents는 state 변숫값을 변경해주는 함수다.

6~8: useEffect() 함수는 생명주기 함수 componentDidMount()와 같이 return되는 html 코드들이 화면에 그려진 이후에 실행된다. 최초 페이지가 로딩될 때 한 번 실행되고 setContents() 함수로 state 값이 변경되면 한 번 더 실행된다.

13: 버튼을 클릭하면, line 4에서 선언한 setContents() 함수로 contents 값을 수정한다. 이때 state 값이 변경되면 화면을 다시 그려주는데, ‘THIS IS REACT’라는 글자가 ‘THIS IS HOOK’로 변경되는 것을 확인할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 44>** |

032. Fragments 사용하기

**• 학습 내용:** Fragments을 사용하는 방법을 이해한다.

**• 힌트 내용:** 컴포넌트에서 element를 return할 때 사용되는 Fragments 기능을 확인한다.

컴포넌트 단위로 element를 return할 때 하나의 <html> 태그로 전체를 감싸지 않으면 에러가 발생한다. 이때 <Fragments> 태그로 감싸면 불필요한 <html> 태그를 추가하지 않고 사용할 수 있다.

|  |
| --- |
| NOTE |
| element 반환 시 하나의 <html> 태그로 감싸지 않으면 다음과 같은 에러 메시지가 표시된다. Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import './App.css';  import Fragments from './R032\_Fragments'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Fragments/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R032\_Fragments.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R032\_Fragments.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R032\_Fragments.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14 | import React, { Component } from 'react';  class R032\_Fragments extends Component {  render() {  return (  <React.Fragment>  <p>P TAG</p>  <span>SPAN TAG</span>  </React.Fragment>  )  }  }  export default R032\_Fragments; |

6~8: <React.Fragment> 태그를 사용하지 않았다면, <p> 태그와 <span> 태그가 하나의 태그로 감싸져 있지 않기 때문에 에러가 발생한다.

|  |
| --- |
| NOTE |
| <Fragment> 태그를 약식으로 다음과 같이 사용할 수 있다.  <>  <p>P TAG</p>  <span>SPAN TAG</span>  </> |

<실행 결과>

|  |
| --- |
| **<그림 45>** |

033. map()으로 element 반환하기

**• 학습 내용:** map() 함수를 사용해 element를 return하는 방법을 이해한다.

**• 힌트 내용:** element들을 map() 함수로 나열해 화면 출력 결과를 확인한다.

반복해서 출력해야 하는 element들을 배열에 넣어두고 map() 함수로 순서대로 나열해 컴포넌트를 return할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from ‘react’;  import ReturnMap from ‘./R033\_ReturnMap’  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReturnMap/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R033\_ReturnMap.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R033\_ReturnMap.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R033\_ReturnMap.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18 | import React, { Component } from ‘react’;  class R033\_ReturnMap extends Component {  render() {  const element\_Array = [  <li>react</li>  , <li>200</li>  , <li>Array map</li>  ]  return (  <ul>  {element\_Array.map((array\_val) => array\_val)}  </ul>  )  }  }  export default R033\_ReturnMap; |

5~9: <li> 태그 리스트를 element\_Array 배열에 순서대로 저장한다.

11~13: map() 함수로 element\_Array 배열에 있는 element들을 순서대로 꺼내 <ul> 태그 안쪽에 출력한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 46>**

034. reactstrap Alerts 사용하기

**• 학습 내용:** reactstrap Alerts 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Alerts 패키지의 기능과 종류를 확인한다.

bootstrap은 프론트엔드 디자인을 쉽게 구현할 수 있도록 도와주는 html, css, js 프레임워크다. bootstrap을 react에서 사용하기 적합하게 패키지로 만든 것이 reactstrap이다. Alerts 패키지는 알림 영역에 대한 기능을 제공한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install –save reactstrap]를 입력하면 다음과 같이 npm이 reactstrap을 설치한다.

|  |
| --- |
|  |

**<그림 48>**

reactstrap은 bootstrap css를 포함하고 있지 않기 때문에 [npm install –save bootstrap]을 입력해bootstrap도 npm으로 설치해준다.

|  |
| --- |
|  |

**<그림 49>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapAlerts from './R034\_ReactstrapAlerts'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapAlerts/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R034\_ReactstrapAlerts.js 파일을 임포트해 사용할 수 있도록 한다.

4: bootstrap.css를 임포트해 <reactstrap> 태그를 사용할 때 bootstrap css가 적용될 수 있도록 한다.

src 폴더에 R034\_ReactstrapAlerts.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R034\_ReactstrapAlerts.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  import { Alert, UncontrolledAlert } from 'reactstrap';  class R034\_ReactstrapAlerts extends Component {  render() {  return (  <div>  <Alert color="light">  Simple Alert [color : light]  </Alert>  <UncontrolledAlert color="warning">  Uncontrolled Alert [color : warning]  </UncontrolledAlert>  </div>  )  }  }  export default R034\_ReactstrapAlerts; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. 기본 알림 Alert과 삭제 기능이 추가된 알림 UncontrolledAlert을 사용하기 위해 선언한다.

8~10: 기본 알림 영역 Alert을 구현한다. 단순히 알림 내용을 표시시켜준다.

11~13: 삭제 가능한 알림 영역 UncontrolledAlert을 구현한다. [x] 버튼을 누르면 알림 영역이 삭제된다.

|  |
| --- |
| NOTE |
| reactstrap에서 color 속성을 primary, secondary, success, danger, warning, info, light, dark 중에 선택하면 각각 다른 색을 적용할 수 있다. |

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 47>**

035. reactstrap Badge 사용하기

**• 학습 내용:** reactstrap Badge 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Badge 패키지의 기능과 종류를 확인한다.

Badge 패키지는 부모 요소에 추가로 특정 문자열이나 숫자를 표시할 때 사용된다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapBadges from './R035\_ReactstrapBadges'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapBadges/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R035\_ReactstrapBadge.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R035\_ReactstrapBadge.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R035\_ReactstrapBadge.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  import { Badge, Button } from 'reactstrap';  class R035\_ReactstrapBadges extends Component {  render() {  return (  <div>  <h1>PRODUCT NAME <Badge color="secondary">hit</Badge></h1>  <Button color="light" outline>  Accessor <Badge color="dark">4</Badge>  </Button>  <Badge color="danger" pill>pill</Badge>  <Badge href="http://example.com" color="light">GoLink</Badge>  </div>  )  }  }  export default R035\_ReactstrapBadges; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Badge와 Button을 사용하기 위해 선언한다.

8: 텍스트 옆에 배지를 추가했다. 사용 예시로 쇼핑몰 웹 사이트의 상품 옆에 신상품(new), 히트 상품(hit), 세일 상품(sale)과 같은 배지를 사용할 수 있다.

9~11: 버튼 안에 배지를 추가했다. 접속자, 메시지의 수와 같은 어떤 속성의 수치를 나타낼 때 사용할 수 있다.

12: pill 속성을 추가하면 테두리를 둥글게 적용할 수 있다.

13: href 속성을 추가해 배지에 링크를 연결할 수 있다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 50>**

036. reactstrap Breadcrumbs 사용하기

**• 학습 내용:** reactstrap Breadcrumbs 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Breadcrumbs 패키지의 기능과 종류를 확인한다.

Breadcrumbs 패키지는 페이지 위치 경로를 지정한 웹 내비게이션에 사용된다. 보통 웹 사이트 상단에 표시되는 메뉴 리스트에 사용하며 특정 메뉴를 선택하면 해당하는 페이지 위치로 이동시킨다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapBreadcrumbs from './R036\_ReactstrapBreadcrumbs'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapBreadcrumbs/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R036\_ReactstrapBreadcrumbs.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R036\_ReactstrapBreadcrumbs.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R036\_ReactstrapBreadcrumbs.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import React, { Component } from 'react';  import { Breadcrumb, BreadcrumbItem } from 'reactstrap';  class R036\_ReactstrapBreadcrumbs extends Component {  render() {  return (  <div id="top">  <Breadcrumb tag="nav" listTag="div">  <BreadcrumbItem tag="a" href="#top">Go\_top</BreadcrumbItem>  <BreadcrumbItem tag="a" href="#bottom">Go\_bottom</BreadcrumbItem>  </Breadcrumb>  <div id="bottom" style={{marginTop:"1000px"}}>  <span>bottom</span>  </div>  </div>  )  }  }  export default R036\_ReactstrapBreadcrumbs; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Breadcrumb와 BreadcrumbItem을 사용하기 위해 선언한다.

8: <nav> 태그와 <div> 태그를 생성한다. bootstrap 코드로는 함축적으로 표현돼 있지만, 개발자 도구에서 elements를 확인해보면 다음과 같이 html의 <nav> 태그와 <div> 태그가 생성된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 53>**

9~10: <a> 태그를 생성하고 href 속성 값으로 이동할 링크의 주소를 넣는다. 예제와 같이 top, bottom이라는 태그의 id 값을 사용하거나 <http://example.com>와 같은 웹 링크를 사용할 수도 있다. Go\_top 링크를 클릭하면 line 7의 <div> 태그로, Go\_bottom 링크를 클릭하면 line 12의 <div> 태그로 화면이 이동한다.

크롬 웹 브라우저에서 서버를 구동하면 다음과 같은 화면을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 51> <그림 52>**

037. reactstrap Button Dropdown 사용하기

**• 학습 내용:** reactstrap Dropdown 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Dropdown 패키지의 기능과 종류를 확인한다.

Dropdown 패키지는 대표 메뉴를 클릭하면, 하위 메뉴 리스트가 표시되는 기능이다. 이때 대표 메뉴를 누를 때마다 하위 메뉴 리스트를 번갈아가며 표시, 미표시을 해야 한다. 그러기 위해서는 현재 하위 메뉴 리스트 상태가 표시인지 미표시인지에 대한 정보가 필요하다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapDropdown from './R037\_ReactstrapDropdown'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapDropdown/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R037\_ReactstrapDropdown.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R037\_ReactstrapDropdown.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R037\_ReactstrapDropdown.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import React, { Component } from 'react';  import { ButtonDropdown, DropdownToggle, DropdownMenu, DropdownItem } from 'reactstrap';  class R037\_ReactstrapDropdown extends Component {  constructor (props) {  super(props);  this.state = {  dropdownOpen: false  }  }  toggle = (e) => {  this.setState({ dropdownOpen: !this.state.dropdownOpen });  }  render() {  return (  <ButtonDropdown isOpen={this.state.dropdownOpen} toggle={this.toggle}>  <DropdownToggle caret> 버튼 Dropdown</DropdownToggle>  <DropdownMenu>  <DropdownItem header>헤더</DropdownItem>  <DropdownItem disabled>비활성화 버튼</DropdownItem>  <a href="http://example.com/">  <DropdownItem>example 웹 사이트로 이동</DropdownItem>  </a>  <DropdownItem onClick={e => alert("Alert 버튼")}>Alert 버튼</DropdownItem>  </DropdownMenu>  </ButtonDropdown>  )  }  }  export default R037\_ReactstrapDropdown; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. ButtonDropdown, DropdownToggle, DropdownMenu, DropdownItem을 사용하기 위해 선언한다.

8: line 18에서 isOpen 값이 true면 하위 메뉴가 표시, false면 미표시된다. 초깃값은 false로 저장한다.

12~14: line 18에서 버튼 영역을 클릭하면, toggle 함수가 실행된다. 이때 하위 메뉴 표시 상태 값을 반대로(true면 false, false면 true로 변경) 만들어준다.

21: 헤더 영역에는 하위 메뉴 리스트에 대한 대표 정보를 기입할 수 있다.

22: disabled 속성을 추가하면, 버튼을 클릭해도 반응하지 않는다.

23~25: <a> 태그로 버튼을 감싸면, 버튼을 클릭했을 때 원하는 url로 이동할 수 있다.

26: onClick 이벤트로 버튼 클릭 시 alert()과 같은 함수를 실행할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 54>**

038. reactstrap Button Group 사용하기

**• 학습 내용:** reactstrap Button Group 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Button Group 패키지의 기능과 종류를 확인한다.

Button Group 패키지는 비슷한 형태와 기능을 하는 버튼들을 그룹으로 관리할 수 있게 지원해준다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapButtonGroup from './R038\_ReactstrapButtonGroup'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapButtonGroup/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R038\_ReactstrapButtonGroup.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R038\_ReactstrapButtonGroup.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R038\_ReactstrapButtonGroup.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | import React, { Component } from 'react';  import { Button, ButtonGroup } from 'reactstrap';  class R038\_ReactstrapButtonGroup extends Component {  constructor (props) {  super(props);  this.state = { number: 10 }  }  move = (type, e) => {  if(type === 'Left'){  this.setState({number: this.state.number - 1})  }else if(type === 'Right')(  this.setState({number: this.state.number + 1})  )  }  render() {  return (  <div style={{padding: "0px"}}>  <ButtonGroup style={{padding: "0px"}}>  <Button onClick={e => this.move('Left')}>Left</Button>  <Button onClick={e => this.move('Middle')}>Middle</Button>  <Button onClick={e => this.move('Right')}>Right</Button>  </ButtonGroup>  <br/>{this.state.number}  </div>  )  }  }  export default R038\_ReactstrapButtonGroup; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Button, ButtonGroup을 사용하기 위해 선언한다.

21~25: 비슷한 기능을 하는 버튼들을 <ButtonGroup> 태그로 묶어 사용한다. 각각의 <Button> 태그를 누를 때마다 move() 함수를 호출한다.

10~16: move() 함수는 파라미터로 받은 type 값으로 버튼을 구분해 state 변수를 변경한다. Left 버튼은 현재 숫자에서 1을 빼고 Right 버튼은 1을 더한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 55>**

039. reactstrap Bottons 사용하기

**• 학습 내용:** reactstrap Bottons 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Bottons 패키지의 기능과 종류를 확인한다.

Bottons 패키지는 <Button> 태그에 color 속성에 약속된 문자열을 넣으면, 용도에 맞는 버튼 스타일을 지원해준다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import './App.css';  import ReactstrapBottons from './R039\_ReactstrapBottons'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapBottons/>  </div>  );  }  export default App; |

3: App.js 파일과 같은 경로에 위치한 R039\_ReactstrapBottons.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R039\_ReactstrapBottons.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R039\_ReactstrapBottons.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React, { Component } from 'react';  import { Button } from 'reactstrap';  class R039\_ReactstrapBottons extends Component {  render() {  return (  <div style={{padding: "0px"}}>  <Button color="primary">blue</Button>  <Button color="info">teal</Button>  <Button color="success">green</Button>  <Button color="warning">yellow</Button>  <Button color="danger">red</Button>  <Button color="dark">dark gray</Button>  <Button color="secondary">gray</Button>  <Button color="light">white</Button>  </div>  )  }  }  export default R039\_ReactstrapBottons; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Button을 사용하기 위해 선언한다.

8~15: color 속성에 약속된 문자열을 넣으면, 각각 다른 색의 버튼이 화면에 표시된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 56>**

040. reactstrap Card 사용하기

**• 학습 내용:** reactstrap Card 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Card 패키지의 기능과 종류를 확인한다.

Card 패키지는 이미지 제목, 부제목, 내용, 버튼 등을 한 세트로 묶는다. Card 단위로 리스트를 만들어 반복해서 출력하면, 정형화된 콘텐츠 목록을 만들 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React from 'react';  import ReactstrapCard from './R040\_ReactstrapCard'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapCard/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R040\_ReactstrapCard.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R040\_ReactstrapCard.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R040\_ReactstrapCard.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import React, { Component } from 'react';  import {  Card, CardImg, CardText, CardBody,  CardTitle, CardSubtitle, Button  } from 'reactstrap';  class R040\_ReactstrapCard extends Component {  render() {  return (  <div>  <Card>  <CardImg top height="200px" src="http://bitly.kr/4KkfRxZfR" alt="Card image" />  <CardBody>  <CardTitle>Card 제목</CardTitle>  <CardSubtitle>Card 부제목</CardSubtitle>  <CardText>Card 내용 Lorem Ipsum is simply dummy text.</CardText>  <Button> 버튼</Button>  </CardBody>  </Card>  </div>  )  }  }  export default R040\_ReactstrapCard; |

3~4: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Card, CardImg, CardText, CardBody, CardTitle, CardSubtitle, Button을 사용하기 위해 선언한다.

11~19: <Card> 태그로 하위 태그들을 묶어준다. <CardImg> 태그는 <img> 태그와 같이 이미지 크기와 url을 지정할 수 있다. <CardBody> 태그로 Card의 제목, 부제목, 내용, 버튼 태그를 묶어 이미지 하단에 표시한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 57>**

041. reactstrap Carousel 사용하기

**• 학습 내용:** reactstrap Carousel 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Carousel 패키지의 기능과 종류를 확인한다.

Carousel 패키지는 슬라이드를 자동으로 회전시키는 기능을 제공한다. 이미지 텍스트, 버튼 등으로 이뤄진 슬라이드를 일정 시간 단위로 다음 슬라이드로 이동시킨다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import ReactstrapCarousel from './R041\_ReactstrapCarousel'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapCarousel/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R041\_ReactstrapCarousel.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R041\_ReactstrapCarousel.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R041\_ReactstrapCarousel.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | import React, { Component } from 'react';  import { UncontrolledCarousel } from 'reactstrap';  const items = [  {  src: 'http://bitly.kr/QUnjzAodA',  altText: ' 슬라이드1 이미지 대체 문구',  caption: ' 슬라이드1 설명',  header: ' 슬라이드1 제목'  },  {  src: 'http://bitly.kr/n3vBOWt9g',  altText: ' 슬라이드2 이미지 대체 문구',  caption: ' 슬라이드2 설명',  header: ' 슬라이드2 제목'  },  {  src: 'http://bitly.kr/uzWLMeIJh',  altText: ' 슬라이드3 이미지 대체 문구',  caption: ' 슬라이드3 설명',  header: ' 슬라이드3 제목'  }  ];  class R041\_ReactstrapCarousel extends Component {  render() {  return (  <UncontrolledCarousel items={items} />  )}  }  export default R041\_ReactstrapCarousel; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. UncontrolledCarousel을 사용하기 위해 선언한다. UncontrolledCarousel은 태그나 함수를 생략해도 기본 기능을 제공해준다. 그래서 기본적인 기능만 사용할 것이라면, Carousel보다 UncontrolledCarousel를 사용하는 것이 간편하다.

4~23: UncontrolledCarousel 슬라이드에서 사용할 데이터를 배열로 만든다. src 값은 슬라이드 이미지 altText는 슬라이드 이미지가 없을 경우 대체 문구, header는 슬라이드 제목, caption은 header 하단에 표시되는 슬라이드 설명이다.

28: <UncontrolledCarousel> 태그에 배열 형태의 데이터를 넣어주면, reactstrap이 Carousel 기능에 필요한 <html> 태그를 그려주고 스타일(css)과 이벤트(function)를 적용한다.

한 슬라이드에 머무르는 시간은 기본값 5초가 적용된다. 웹 브라우저 가로 너비가 특정 값보다 작으면, 슬라이드 제목과 설명 부분이 표시되지 않을 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 58>**

042. reactstrap Collapse 사용하기

**• 학습 내용:** reactstrap Collapse 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Collapse 패키지의 기능과 종류를 확인한다.

Collapse 패키지는 특정 영역을 펼치고 숨기는 기능을 제공한다. 버튼 이벤트로 펼치고 숨기는 기능을 제어할 수 있다. 상단부터 점차적으로 펼쳐지고 하단부터 숨겨진다는 것이 show(), hide() 함수와의 차이점이다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import ReactstrapCollapse from './R042\_ReactstrapCollapse'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapCollapse/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R042\_ReactstrapCollapse.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R042\_ReactstrapCollapse.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R042\_ReactstrapCollapse.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import { UncontrolledCollapse, Button, CardBody, Card } from 'reactstrap';  class R042\_ReactstrapCollapse extends Component {  render() {  return (  <div>  <Button color="warning" id="toggle">  펼치기/접기  </Button>  <UncontrolledCollapse toggler="#toggle">  <Card>  <CardBody>  REACT 200  </CardBody>  </Card>  </UncontrolledCollapse>  </div>  )}  }  export default R042\_ReactstrapCollapse; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. UncontrolledCollapse, Button, CardBody, Card을 사용하기 위해 선언한다. UncontrolledCollapse은 태그나 함수를 생략해도 기본 기능을 제공해준다. 그래서 기본적인 기능만 사용할 것이라면, Collapse보다 UncontrolledCollapse를 사용하는 것이 간편하다.

8~10: line 11에 toggler 속성에서 #을 사용해 toggle 함수를 호출하는 id 값(toggle)을 지정했다. 버튼을 클릭하면 UncontrolledCollapse 패키지 내부적으로 선언된 toggle 함수(펼치기/숨기기 함수)를 실행한다.

11~17: <UncontrolledCollapse> 태그 toggler 속성으로 버튼 id를 지정해주고 태그 안에 화면에 표시시킬 내용을 작성한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 59>**

0**43**. reactstrap Fade 사용하기

**• 학습 내용:** reactstrap Fade 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Fade 패키지의 기능과 종류를 확인한다.

Fade 패키지는 특정 영역을 서서히 나타내고 숨기는 기능을 제공한다. 버튼 이벤트로 기능을 제어할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import ReactstrapFade from './R043\_ReactstrapFade'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapFade/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R043\_ReactstrapFade.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R043\_ReactstrapFade.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R043\_ReactstrapFade.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import React, { Component } from 'react';  import { Button, Fade } from 'reactstrap';  class R043\_ReactstrapFade extends Component {  constructor (props) {  super(props);  this.state = { fadeInOut: true }  }  toggle = (e) => {  this.setState({ fadeInOut: !this.state.fadeInOut })  }  render() {  return (  <div>  <Button color="success" onClick={this.toggle}>Fade In/Out</Button>  <Fade in={this.state.fadeInOut} tag="h1">  Lorem ipsum dolor sit amet, consectetur adipiscing elit  </Fade>  </div>  )}  }  export default R043\_ReactstrapFade; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Button, Fade을 사용하기 위해 선언한다.

7: <Fade> 태그 영역의 표시 상태를 state로 선언하고 초깃값은 표시인 true로 할당한다.

18~20: <Fade> 태그는 in 속성이 true면 표시, false면 미표시된다. fadeInOut의 초깃값이 true이기 때문에 최초 화면 로딩 시 <Fade> 태그 영역이 표시된다.

17: 버튼을 누르면 toggle 함수가 실행된다.

10~12: toggle 함수는 fadeInOut 값을 현재 상태의 반대로 할당한다. setState() 함수가 실행되면 render() 함수가 한 번 더 실행돼 변경된 fadeInOut 값이 적용된다. 결과적으로 버튼이 클릭될 때마다 <Fade> 태그 영역은 현재 상태가 표시일 경우 미표시, 미표시일 경우 표시된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 60>**

044. reactstrap Form 사용하기

**• 학습 내용:** reactstrap Form 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Form 패키지의 기능과 종류를 확인한다.

Form 패키지를 이용하면 기존 <html form> 태그에 깔끔하고 정리된 스타일을 간편하게 적용해 사용할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import ReactstrapForm from './R044\_ReactstrapForm'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ReactstrapForm/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R044\_ReactstrapForm.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R044\_ReactstrapForm.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R044\_ReactstrapForm.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | import React, { Component } from 'react';  import { Form, Label, Input, Row, Col, FormGroup } from 'reactstrap';  class R044\_ReactstrapForm extends Component {  render() {  return (  <Form>  <Label for="exampleGender">gender</Label>  <Input type="select" bsSize="sm">  <option>no select</option>  <option>woman</option>  <option>man</option>  </Input>  <Row form>  <Col md={6}>  <FormGroup>  <Label for="exampleAddress">address</Label>  <Input type="text" name="address" id="address"/>  </FormGroup>  </Col>  <Col md={4}>  <FormGroup>  <Label for="exampleMobile">mobile</Label>  <Input type="text" name="mobile" id="mobile"/>  </FormGroup>  </Col>  <Col md={2}>  <FormGroup>  <Label for="exampleAge">age</Label>  <Input type="text" name="age" id="age"/>  </FormGroup>  </Col>  </Row>  </Form>  )}  }  export default R044\_ReactstrapForm; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Form, Label, Input, Row, Col, FormGroup을 사용하기 위해 선언한다.

7~34: <Form> 태그를 이용해 하나의 폼 단위로 사용할 영역을 묶어준다.

8: <Label> 태그로 폼 요소를 설명하는 문자열을 추가할 수 있다.

9~13: input 타입을 select로 입력해 <option> 태그 중 하나를 선택할 수 있다. 이때 bsSize 속성으로 태그의 크기를 정할 수 있다. sm대신 lg를 사용하면 큰 크기의 선택 박스를 사용할 수 있다.

14~33: <Row> 태그는 줄 단위로 태그들을 묶어 사용할 수 있게 해준다. <Col> 태그는 <Row> 태그로 묶인 영역들이 각각 어느 정도의 영역을 차지할지 정해준다. <Col> 태그의 md 속성으로 너비 값을 정할 수 있는데, 한 줄에 12개의 영역으로 나뉜다. 한 <Row> 태그에 속한 <Col> 태그의 md 속성의 합이 12가 되도록 해야 한다. <FormGrop> 태그는 비슷한 용도의 태그들을 <div> 태그로 감싸 사용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 61>**

045. reactstrap Input Group 사용하기

**• 학습 내용:** reactstrap Input Group 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Input Group 패키지의 기능과 종류를 확인한다.

Input Group 패키지는 여러 개의 태그를 하나의 input 그룹으로 묶어서 사용할 수 있도록 지원한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import InputGroup from './R045\_ReactstrapInputGroup'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <InputGroup/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R045\_ReactstrapInputGroup.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R045\_ReactstrapInputGroup.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R045\_ReactstrapInputGroup.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import { InputGroup, InputGroupAddon, InputGroupText, Input, Button } from 'reactstrap';  class R045\_ReactstrapInputGroup extends Component {  render() {  return (  <>  <InputGroup>  <Input placeholder="userid" />  <InputGroupAddon addonType="append">  <InputGroupText>@reactmail.com</InputGroupText>  </InputGroupAddon>  </InputGroup>  <InputGroup>  <InputGroupAddon addonType="prepend"><Button> 버튼</Button></InputGroupAddon>  <Input />  </InputGroup>  </>  )}  }  export default R045\_ReactstrapInputGroup; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. InputGroup, InputGroupAddon, InputGroupText, Input, Button을 사용하기 위해 선언한다.

8~13: <InputGroup> 태그로 <input> 태그와 고정 텍스트 표시 영역(@reactmail.com)을 묶어 사용한다. <InputGroupAddon> 태그의 addonType 속성 값을 append로 하면, <input> 태그의 오른쪽에 표시하기 적합한 스타일이 적용된다.

14~17: <InputGroup> 태그로 <input> 태그와 버튼 영역을 묶어 사용한다. <InputGroupAddon> 태그의 addonType 속성 값을 prepend로 하면, <input> 태그의 왼쪽에 표시하기 적합한 스타일이 적용된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 62>**

046. reactstrap Jumbotron 사용하기

**• 학습 내용:** reactstrap Jumbotron 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Jumbotron 패키지의 기능과 종류를 확인한다.

Jumbotron은 대형 전광판을 의미한다. 넓은 영역에 눈에 띄게 정보를 표시해 사용자의 관심을 불러 일으킬 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import Jumbotron from './R046\_ReactstrapJumbotron'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <Jumbotron/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R046\_ReactstrapJumbotron.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R046\_ReactstrapJumbotron.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R046\_ReactstrapJumbotron.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React, { Component } from 'react';  import { Jumbotron, Button } from 'reactstrap';  class R046\_ReactstrapJumbotron extends Component {  render() {  return (  <>  <Jumbotron style={{backgroundColor: "#D38C7C"}}>  <h1 className="display-4">REACT 200</h1>  <p className="h4">Contrary to popular belief, Lorem Ipsum is not simply random text.</p>  <hr className="my-2" />  <p>There are many variations of passages of Lorem Ipsum available.</p>  <p className="lead">  <Button color="danger">Go Detail</Button>  </p>  </Jumbotron>  </>  )}  }  export default R046\_ReactstrapJumbotron; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Jumbotron, Button을 사용하기 위해 선언한다.

8~13: <Jumbotron> 태그로 하위 태그들을 묶어주면, 넓은 영역을 차지하는 <div> 태그가 생성된다. <Jumbotron> 태그에 배경색, 배경 이미지 등 추가 속성을 적용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 63>**

047. reactstrap List Group 사용하기

**• 학습 내용:** reactstrap List Group 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap List Group 패키지의 기능과 종류를 확인한다.

List Group 패키지는 정돈된 스타일의 목록을 표시할 때 사용한다. <ListGroup>과 <ListGroupItem> 태그는 html에서 목록을 만드는 태그인 <ul>, <li>로 변환된다. Button과 링크(<a> 태그)를 별도의 태그를 추가하지 않고 속성 값으로 간편하게 사용할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from 'react';  import ListGroup from './R047\_ReactstrapListGroup'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <p>CSS 적용하기</p>  <ListGroup/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R047\_ReactstrapListGroup.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R047\_ReactstrapListGroup.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R047\_ReactstrapListGroup.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React, { Component } from 'react';  import { ListGroup, ListGroupItem, Badge } from 'reactstrap';  class R047\_ReactstrapListGroup extends Component {  com  render() {  return (  <>  <ListGroup>  <ListGroupItem color="danger" className="justify-content-between">Badge  <Badge pill>200</Badge>  </ListGroupItem>  <ListGroupItem disabled tag="a" href="#">Disable</ListGroupItem>  <ListGroupItem tag="a" href="http://example.com">Link</ListGroupItem>  <ListGroupItem tag="button" action onClick={e => alert("button")}>Button</ListGroupItem>  </ListGroup>  </>  )}  }  export default R047\_ReactstrapListGroup; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. ListGroup, ListGroupItem, Badge을 사용하기 위해 선언한다.

9~16: 목록으로 사용할 요소들을 <ListGroup> 태그로 묶어준다.

10~12: <ListGroupItem> 태그로 목록의 요소를 추가한다. reactstrap에서 지원하는 color 속성과 <Badge> 태그를 사용해 간편하게 목록 색상과 추가정보를 표현할 수 있다.

13: disabled 속성을 추가하면 목록 요소를 비활성화할 수 있다.

14: tag 속성 값을 a로 하면 <a> 태그가 추가된다. href 속성 값은 <a> 태그와 동일하게 이동할 링크 정보를 넣어준다.

15: tag 속성 값을 button으로 하면 <button> 태그가 추가된다. onClick 이벤트를 추가해 목록 요소를 클릭 시 지정한 함수를 실행한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 64>**

048. reactstrap Modal 사용하기

**• 학습 내용:** reactstrap Modal 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Modal 패키지의 기능과 종류를 확인한다.

Modal 패키지는 alert() 함수와 마찬가지로 사용자에게 원하는 시점에 알림 창을 띄워 필요한 내용을 보여준다. Modal은 alert() 함수와 달리, 웹 브라우저에서 팝업 창을 차단할 수 없고 배경 페이지와 어울리는 디자인을 적용할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import Modal from './R048\_ReactstrapModal'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Modal/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R048\_ReactstrapModal.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R048\_ReactstrapModal.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R048\_ReactstrapModal.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import React, { Component } from 'react';  import { Button, Modal, ModalHeader, ModalBody, ModalFooter } from 'reactstrap';  class R048\_ReactstrapModal extends Component {  constructor(props) {  super(props);  this.state = { modal: false }  }  toggle = () => {  this.setState({modal: !this.state.modal})  }  render() {  return (  <>  <Button color="warning" onClick={this.toggle}>Modal 버튼</Button>  <Modal isOpen={this.state.modal} fade={true} toggle={this.toggle}>  <ModalHeader toggle={this.toggle}>Modal Header</ModalHeader>  <ModalBody>  The generated Lorem Ipsum is therefore always free from repetition.  </ModalBody>  <ModalFooter>  <Button color="primary" onClick={this.toggle}>확인</Button>  <Button color="secondary" onClick={this.toggle}>닫기</Button>  </ModalFooter>  </Modal>  </>  )  }  }  export default R048\_ReactstrapModal; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Button, Modal, ModalHeader, ModalBody, ModalFooter을 사용하기 위해 선언한다.

7: Modal 팝업의 표시 상태를 state 변수로 선언하고 초깃값으로 미표시(false)를 할당한다.

17: 버튼을 누르면, toggle 함수가 실행되고 line 11에서 Modal 팝업 표시 상태를 현재와 반대로 변경한다.

18~27: 팝업 창으로 표시할 영역을 <Modal> 태그로 묶는다. isOpen 속성이 true면 표시, false면 미표시되는데, line 11에서 state 변수 modal의 상태가 변경되는 것에 따라 동작한다. fade 속성을 true로 하면 팝업 창이 서서히 표시시킬 수 있다. toggle 속성에 입력된 toggle 함수는 팝업 이외의 영역을 클릭했을 때 실행된다.

19: <ModalHeader> 태그는 팝업에서 상단 헤더 영역을 나타낸다. toggle 속성에 입력된 toggle 함수는 헤더 오른쪽 영역의 [x]를 클릭했을 때 실행된다.

20~22: <ModalBody> 태그는 팝업 내용이 표시되는 영역이다.

23~26: <ModalFooter> 태그는 팝업 하단 영역으로, 팝업 창에 필요한 버튼을 추가할 수 있다. 버튼 이벤트로 필요한 함수를 연결해 사용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 65>**

049. reactstrap Navbar 사용하기

**• 학습 내용:** reactstrap Navbar 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Navbar 패키지의 기능과 종류를 확인한다.

Navbar 패키지는 웹 사이트의 내부 페이지들로 쉽게 이동할 수 있도록 메뉴 리스트와 링크를 제공한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import Navbar from './R049\_ReactstrapNavbar'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Navbar/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R049\_ReactstrapNavbar.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R049\_ReactstrapNavbar.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R049\_ReactstrapNavbar.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | import React, { Component } from 'react';  import { Collapse, Navbar, NavbarToggler, NavbarBrand, Nav, NavItem, NavLink } from 'reactstrap';  class R048\_ReactstrapModal extends Component {  constructor(props) {  super(props);  this.state = { collapsed: false }  }  toggle = () => {  this.setState({collapsed: !this.state.collapsed})  }  render() {  return (  <>  <Navbar color="faded" light>  <NavbarBrand href="/" className="mr-auto">Navbar</NavbarBrand>  <NavbarToggler onClick={this.toggle} className="mr-2" />  <Collapse isOpen={this.state.collapsed} navbar>  <Nav navbar>  <NavItem>  <NavLink href="#">react</NavLink>  </NavItem>  <NavItem>  <NavLink href="http://example.com/">200</NavLink>  </NavItem>  </Nav>  </Collapse>  </Navbar>  </>  )  }  }  export default R048\_ReactstrapModal; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Collapse, Navbar, NavbarToggler, NavbarBrand, Nav, NavItem, NavLink을 사용하기 위해 선언한다.

7: Navbar리스트의 표시 상태를 state 변수로 선언하고 초깃값으로 미표시(false)를 할당한다.

17~30: 전체 내비게이션 영역을 <Navbar> 태그로 묶는다.

18: <NavbarBrand> 태그는 내비게이션에서 무조건 표시되는 영역이다.

19: <NavbarToggler> 태그는 화면 오른쪽에 표시되는 버튼으로, 클릭 시 toggle 함수를 실행한다.

20: <Collapse> 태그는 line 19에서 toggle 함수가 실행되면, 표시 또는 미표시되는 영역이다. toggle 함수에서 변경되는 state 변수, collapsed 값에 따라 동작한다.

21~28: Nav 안에 표시할 메뉴 리스트를 <NavItem> 태그로 나열한다. <NavItem> 태그 안에 NavLink를 추가하고 메뉴명과 링크를 추가할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 66>**

050. reactstrap Pagination 사용하기

**• 학습 내용:** reactstrap Pagination 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Pagination 패키지의 기능과 종류를 확인한다.

데이터 수가 많아 한 페이지에 모두 표시할 수 없을 때 여러 페이지에 나눠 표시한다. Pagination 패키지는 페이지 번호, 이전/다음 페이지 첫/마지막 페이지 버튼을 쉽게 구현할 수 있게 지원한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import Pagination from './R050\_ReactstrapPagination'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Pagination/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R050\_ReactstrapPagination.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R050\_ReactstrapPagination.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R050\_ReactstrapPagination.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  import { Pagination, PaginationItem, PaginationLink } from 'reactstrap';  class R048\_ReactstrapModal extends Component {  pagination = (type) => {  alert("Go "+type)  }  render() {  return (  <>  <Pagination size="lg" aria-label="Page navigation example">  <PaginationItem>  <PaginationLink previous onClick={e => this.pagination("previous")} />  </PaginationItem>  <PaginationItem>  <PaginationLink onClick={e => this.pagination("1")}>1</PaginationLink>  </PaginationItem>  <PaginationItem>  <PaginationLink onClick={e => this.pagination("2")}>2</PaginationLink>  </PaginationItem>  <PaginationItem>  <PaginationLink last onClick={e => this.pagination("last")} />  </PaginationItem>  </Pagination>  </>  )  }  }  export default R048\_ReactstrapModal; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Pagination, PaginationItem, PaginationLink을 사용하기 위해 선언한다.

5~7: 페이지 버튼을 클릭했을 때 파라미터에 따라 알림 창을 표시한다.

12~25: <Pagination> 태그로 전체 페이징에 사용할 영역을 묶어준다. <PaginationItem> 태그 단위로 각각의 버튼을 구분한다. <PaginationLink> 태그로 버튼에 표시될 텍스트를 감싼다. 이때 텍스트를 감싸는 대신 속성 값(first, previous, next, last)을 추가해 표현할 수 있다. onClick 이벤트를 추가하면 함수를 실행해 필요한 데이터 리스트를 호출할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 67>**

051. reactstrap Popovers 사용하기

**• 학습 내용:** reactstrap Popover 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Popover 패키지의 기능과 종류를 확인한다.

Popover 패키지는 html 요소를 클릭했을 때 요소에 연결된 메시지 박스를 띄울 수 있는 기능을 제공한다. tooltip과 거의 동일하게 동작한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import Popover from './R051\_ReactstrapPopover'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Popover/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R051\_ReactstrapPopover.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R051\_ReactstrapPopover.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R051\_ReactstrapPopover.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import { Button, UncontrolledPopover, PopoverHeader, PopoverBody } from 'reactstrap';  class R051\_ReactstrapPopover extends Component {  render() {  return (  <>  <Button id="Popover\_id" type="button">  Popover button  </Button>  <UncontrolledPopover placement="right" target="Popover\_id">  <PopoverHeader>React 200</PopoverHeader>  <PopoverBody>Aenean id magna id risus congue ornare.  Vestibulum sed diam et mi pulvinar facilisis sed eu risus.  </PopoverBody>  </UncontrolledPopover>  </>  )  }  }  export default R051\_ReactstrapPopover; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Button, UncontrolledPopover, PopoverHeader, PopoverBody을 사용하기 위해 선언한다. UncontrolledPopover은 태그나 함수를 생략해도 기본 기능을 제공해준다. 그래서 기본적인 기능만 사용할 것이라면, Popover보다 UncontrolledPopover를 사용하는 것이 간편하다.

8~10: <Button> 태그의 id 값을 <UncontrolledPopover> 태그의 target 속성 값과 동일하게 해주면, 버튼을 눌렀을 때 <UncontrolledPopover> 태그 영역이 나타난다. 버튼을 누를 때마다 <UncontrolledPopover> 태그 영역의 표시 상태가 반대로 바뀐다.

11: placement는 요소를 기준으로 메시지 박스가 표시되는 위치 속성이다. top(위), bottom(아래), left(왼쪽), right(오른쪽)을 사용할 수 있다.

12~15: <UncontrolledPopover> 태그 안에서 <PopoverHeader> 태그로 헤더 영역에 제목을 입력할 수 있다. <PopoverBody> 태그에는 메시지 내용을 표시시킬 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 68>**

052. reactstrap Progress 사용하기

**• 학습 내용:** reactstrap Progress 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Progress 패키지의 기능과 종류를 확인한다.

Progress 패키지는 전체 작업에 대한 현재 진행 상태를 표현해준다. 진행 바 내부에 문자열을 넣을 수 있고 색상을 적용할 수도 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import Progress from './R052\_ReactstrapProgress'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Progress/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R052\_ReactstrapProgress.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R052\_ReactstrapProgress.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R052\_ReactstrapProgress.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | import React, { Component } from 'react';  import { Progress } from 'reactstrap';  class R051\_ReactstrapPopover extends Component {  constructor (props) {  super(props);  this.state = { progress:0 }  }  componentDidMount(){ this.progress() }  progress =()=>{  if(this.state.progress !== 100){  setTimeout(function() {  this.setState({progress: this.state.progress+1});  this.progress();  }.bind(this), 100);  }  }  render() {  return (  <>  <Progress color="info" value={this.state.progress}>{this.state.progress}%</Progress><br/>  <Progress multi>  <Progress bar color="warning" value="25">25%</Progress>  <Progress bar color="success" value="35">Wow!</Progress>  <Progress bar value="15">Meh</Progress>  <Progress bar color="danger" value="25">LOOK OUT!!</Progress>  </Progress>  </>  )  }  }  export default R051\_ReactstrapPopover; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Progress를 사용하기 위해 선언한다.

12~19: line 10에서 componentDidMount() 함수가 실행되면서 progress() 함수를 호출한다. setTimeout() 함수가 0.1초마다 실행되는데, 진행 상태 값(progress)을 1씩 증가시켜 setState() 함수로 변경한다.

24: line 15에서 setState() 함수가 실행될 때마다 render() 함수가 실행된다. 이때 {this.state.progress} 값이 1씩 증가하기 때문에 진행 바가 0%에서 100%으로 채워지는 것을 확인할 수 있다.

25~30: 여러 개의 진행 바를 사용할 때는 <Progress multi> 태그로 묶어준다. <Progress> 태그의 속성은 진행 바 형태(bar, animated, striped), 색상(color), 진행률(value)이 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 69>**

053. reactstrap Spinner 사용하기

**• 학습 내용:** reactstrap Spinner 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Spinner 패키지의 기능과 종류를 확인한다.

Spinner 패키지는 어떤 작업이 진행되고 있음을 표시하는 움직이는 원 형태의 디자인을 제공한다. 색상, 너비, 높이 값을 지정할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import ReactstrapSpinner from './R053\_ReactstrapSpinner'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactstrapSpinner/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R053\_ReactstrapSpinner.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R053\_ReactstrapSpinner.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R053\_ReactstrapSpinner.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React, { Component } from 'react';  import { Spinner } from 'reactstrap';  class R053\_ReactstrapSpinner extends Component {  render() {  return (  <>  <Spinner color="secondary" />  <Spinner color="success" />  <Spinner type="grow" color="dark" />  <Spinner type="grow" color="info" />  <Spinner size="sm" color="primary" />  <Spinner style={{ width: '10rem', height: '0.5rem' }} color="dark" /><br/>  <Spinner style={{ width: '10rem', height: '10rem' }} color="secondary" />  <Spinner style={{ width: '3rem', height: '10rem' }} type="grow" color="primary" />  </>  )  }  }  export default R053\_ReactstrapSpinner; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Spinner를 사용하기 위해 선언한다.

8~9: <Spinner> 태그에 color 속성을 추가하면 색상을 변경할 수 있다.

10~11: <Spinner> 태그에 type 속성을 grow로 추가하면, 원이 중앙에서 점점 커지다가 사라지는 디자인이 적용된다.

12: <Spinner> 태그에 size 속성을 추가하면, 약속된 크기로 표시시킬 수 있다. 기본 크기보다 작게 표시하려면 size 값을 sm으로 입력한다.

13~15: <Spinner> 태그에 style 속성 값으로 너비(width)와 높이(height) 값을 할당하면, 원하는 크기로 나타낼 수 있다. rem은 css 크기 단위로, 기본 크기의 몇 배의 크기로 표시할지 정할 수 있다. 너비와 높이를 같은 값으로 하지 않으면, 타원형(line 15)이나 비정상적인(line 13) 모습으로 표시될 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 70>**

054. reactstrap Table 사용하기

**• 학습 내용:** reactstrap Table 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Table 패키지의 기능과 종류를 확인한다.

Table 패키지는 <html table> 태그에 간편하게 스타일을 적용할 수 있도록 지원한다. 테이블 색상, 가로줄, 세로줄, 크기, hover, striped 속성을 지정할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import ReactstrapTable from './R054\_ReactstrapTable'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactstrapTable/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R054\_ReactstrapTable.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R054\_ReactstrapTable.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R054\_ReactstrapTable.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | import React, { Component } from 'react';  import { Table } from 'reactstrap';  class R054\_ReactstrapTable extends Component {  render() {  return (  <Table>  {/\* <Table dark bordered>  <Table striped hover>  <Table borderless size="sm"> \*/}  <thead>  <tr>  <th>number</th>  <th>Book Name</th>  <th>Price</th>  </tr>  </thead>  <tbody>  <tr>  <th scope="row">1</th>  <td>React 100</td>  <td>￦10000</td>  </tr>  <tr>  <th scope="row">2</th>  <td>React 200</td>  <td>￦20000</td>  </tr>  </tbody>  </Table>  )  }  }  export default R054\_ReactstrapTable; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. Table를 사용하기 위해 선언한다.

7: 속성이 추가되지 않은 <Table> 태그의 스타일은 흰색 배경에 가로줄만 있다.

8: line 7~10 중 line 8만 주석을 해제한다. dark 속성으로 배경이 어둡게 변하고 bordered 속성으로 세로줄이 생긴다.

9: line 7~10중 line 9만 주석을 해제한다. striped 속성은 짝수 행에만 음영을 준다. hover 속성은 마우스 커서가 위치한 행에 음영을 주고 마우스 커서가 벗어나면 음영을 제거한다.

10: line 7~10 중 line 10만 주석을 해제한다. borderless 속성은 가로줄, 세로줄 모두를 제거한다. size 속성을 sm으로 하면 테이블 크기가 축소된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 71>**

055. reactstrap Tab 사용하기

**• 학습 내용:** reactstrap Tab 사용 방법을 이해한다.

**• 힌트 내용:** reactstrap Tab 패키지의 기능과 종류를 확인한다.

Tab 패키지는 사용자 동작에 따라 특정 영역에 다른 내용을 표시할 때 사용한다. 클릭 이벤트에 따라 원하는 영역을 표시해야 할 경우, 간편하게 사용할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14 | import React from 'react';  import ReactstrapTab from './R055\_ReactstrapTab'  import 'bootstrap/dist/css/bootstrap.css'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactstrapTab/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R055\_ReactstrapTab.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R055\_ReactstrapTab.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R055\_ReactstrapTab.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import React, { Component } from 'react';  import { TabContent, TabPane, Nav, NavItem, NavLink } from 'reactstrap';  class R055\_ReactstrapTab extends Component {  constructor (props) {  super(props);  this.state = { TabState:'React' }  }  toggle = (tabnum) => {  if(this.state.TabState !== tabnum) this.setState({TabState:tabnum});  }  render() {  return (  <>  <Nav tabs>  <NavItem>  <NavLink onClick={() => { this.toggle('React'); }}>First Tab</NavLink>  </NavItem>  <NavItem>  <NavLink onClick={() => { this.toggle('200'); }}>Second Tab</NavLink>  </NavItem>  </Nav>  <TabContent activeTab={this.state.TabState}>  <TabPane tabId="React"><h3>React</h3></TabPane>  <TabPane tabId="200"><h3>200</h3></TabPane>  </TabContent>  </>  )  }  }  export default R055\_ReactstrapTab; |

2: reactstrap 패키지에서 사용할 기능을 {} 안에 나열한다. TabContent, TabPane, Nav, NavItem, NavLink를 사용하기 위해 선언한다.

7: 표시할 tab 영역의 id를 상태 변수로 선언한다. 초깃값으로 첫 번째 탭의 tabid인 React로 할당한다.

19, 22: 내비게이션 메뉴를 클릭하면, toggle 함수를 호출하는데, 파라미터로 표시할 tab 영역의 tabid 값을 넘긴다.

10~12: 파라미터로 넘어온 값(tabnum)이 현재 tabid의 상태(Tabstate) 값과 다르다면, 변경된 값으로 상태(Tabstate) 값을 수정한다.

25~28: line 11에서 변경된 상태(Tabstate) 값을 <TabContent> 태그의 activeTab 속성에 할당한다. 이때 할당된 값과 <TabPane> 태그의 tabId 값과 비교해 일치하는 <TabPane> 태그 영역을 표시한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 72>**

056. sweetalert2 Basic 사용하기

**• 학습 내용:** sweetalert2 Basic 사용 방법을 이해한다.

**• 힌트 내용:** sweetalert2 Basic의 기능과 종류를 확인한다.

sweetalert2 패키지는 다양한 디자인과 기능의 알림 창을 지원한다. 기본 자바스크립트 alert()와 같이 사용자에게 필요한 정보를 알림 창으로 표시한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install sweetalert2]를 입력하면 다음과 같이 npm이 sweetalert2를 설치한다.

|  |
| --- |
|  |

**<그림 74>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Sweetalert2Basic from './R056\_Sweetalert2Basic'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Sweetalert2Basic/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R056\_Sweetalert2Basic.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R056\_Sweetalert2Basic.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R056\_Sweetalert2.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18 | import React, { Component } from 'react';  import Swal from 'sweetalert2'  class R056\_Sweetalert2Basic extends Component {  componentDidMount(){  // Swal.fire('1. SweetAlert')  // alert('2. alert()')  Swal.fire('1. SweetAlert').then(result => { alert('2. result.value : '+result.value) })  }  render() {  return (  <h1>sweetalert2</h1>  )  }  }  export default R056\_Sweetalert2Basic; |

2: sweetalert2에서 지원하는 알림 창을 사용하기 위해 sweetalert2을 임포트한다.

6~7: sweetalert2와 alert()의 차이를 확인하기 위해 line 6, 7을 주석 해제하고 line 8을 주석 처리한다. 코드를 실행하면, line 7이 line 6보다 먼저 실행된다. 이유는 sweetalert2는 비동기적으로 동작해 Swal.fire() 함수를 실행시켜 놓고 완료 여부에 상관없이 다음 코드(line 7)을 실행하기 때문이다. 이때 alert() 함수는 동기적으로 동작하기 때문에 [확인] 버튼을 누르지 않으면 다음 작업으로 넘어가지 않는다.

|  |
| --- |
| NOTE |
| 동기는 요청과 결과가 동시에 일어난다는 뜻이고 비동기는 요청과 결과의 작업 처리 단위를 맞추지 않는다는 의미다. |

8: 비동기적으로 실행되는 sweetalert2 함수를 동기적으로 사용하기 위해서 line 8의 주석을 해제하고 line 6, 7을 주석 처리한다. 프로미스의 then 함수를 사용해 Swal.fire() 함수의 결괏값(result)이 반환됐을 때 alert() 함수를 실행한다. Swal.fire() 함수 알림 창에서 OK 버튼을 누르면 result 값이 true로 반환된다.

|  |
| --- |
| NOTE |
| 프로미스란, 비동기 함수를 동기적으로 사용할 수 있도록 하는 개념이다. 프로미스 함수 중 then은 이전 함수 실행이 완전히 완료됐을 때 실행된다. |

<실행 결과>

|  |
| --- |
|  |

**<그림 73>**

057. sweetalert2 position 사용하기

**• 학습 내용:** sweetalert2 position 사용 방법을 이해한다.

**• 힌트 내용:** sweetalert2 position의 기능과 종류를 확인한다.

sweetalert2 패키지는 알림 창의 표시 방식과 구성 요소들을 원하는 형태로 변경할 수 있는 옵션을 제공한다. 옵션 중 position은 알림 창의 표시 위치를 결정한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Sweetalert2Position from './R057\_Sweetalert2Position'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Sweetalert2Position/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R057\_Sweetalert2Position.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R057\_Sweetalert2Position.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R057\_Sweetalert2Position.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import React, { Component } from 'react';  import Swal from 'sweetalert2'  class R057\_Sweetalert2Position extends Component {  saveAlert = (flag, positionflag, e) => {  Swal.fire({  position: positionflag,  icon: 'success',  title: flag+'됐습니다.',  showConfirmButton: false,  timer: 1500  })  }  render() {  return (  <>  <h1>sweetalert2</h1>  <button onClick={e => this.saveAlert('저장', 'center')}>저장</button>  <button onClick={e => this.saveAlert('수정', 'bottom-end')}>수정</button>  </>  )  }  }  export default R057\_Sweetalert2Position; |

5~12: saveAlert 함수는 파라미터로 두 가지 변수(flag, positionflag)를 받는다. Swal.fire 함수의 옵션 중 title은 알림 창에 표시되는 문구인데, line 9에서 파라미터 flag에 따라 표시 문구가 달라진다. line 7 position은 알림 창의 표시 위치를 정하는 옵션이고 파라미터 positionflag의 값을 할당한다.

|  |
| --- |
| NOTE |
| Swal.fire 함수의 postion 옵션 값은 다음과 같은 형식으로 사용할 수 있다. top : 중앙 상단, top-start : 좌측 상단, top-end : 우측 상단, center : 정중앙, center-start : 중앙 좌측, center-end : 중앙 우측, bottom : 중앙 하단, bottom-start : 좌측 하단, bottom-end : 우측 하단 |

19: [저장] 버튼을 누르면, 페이지 정중앙에 ‘저장되었습니다.’ 알림 창이 표시된다.

20: [저장] 버튼을 누르면, 페이지 우측 하단에 ‘수정되었습니다.’ 알림 창이 표시된다.

<실행 결과>

|  |
| --- |
| **<그림 75>** |

**<그림 76>**

058. sweetalert2 confirm 사용하기

**• 학습 내용:** sweetalert2 confirm 사용 방법을 이해한다.

**• 힌트 내용:** sweetalert2 confirm 패키지의 기능에 대해 확인한다.

웹 사이트에 삭제 기능을 구현할 때 사용자의 실수로 데이터가 삭제될 수 있다. 이런 상황을 방지하기 위해 실제 삭제 작업을 실행하기 전에 다시 한번 확인하는 알림 창을 표시한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Sweetalert2Confirm from './R058\_Sweetalert2Confirm'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Sweetalert2Confirm/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R058\_Sweetalert2Confirm.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R058\_Sweetalert2Confirm.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R058\_Sweetalert2Confirm.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | import React, { Component } from 'react';  import Swal from 'sweetalert2'  class R058\_Sweetalert2Confirm extends Component {  deletAlert = (e) => {  Swal.fire({  title: '정말 삭제하시겠습니까?',  icon: 'question',  showCancelButton: true,  confirmButtonColor: '#4B088A',  cancelButtonColor: '#01DF01',  confirmButtonText: '네',  cancelButtonText: '아니요'  }).then((result) => {  if (result.value) {  document.getElementById('deleteId').remove();  Swal.fire(  'Deleted',  'sweetalert2 삭제완료',  'success'  )  }  })  }  render() {  return (  <>  <h1 id='deleteId'>sweetalert2</h1>  <button onClick={e => this.deletAlert()}>삭제</button>  </>  )  }  }  export default R058\_Sweetalert2Confirm; |

5~14: deletAlert 함수가 실행되면 confirm 알림 창으로, 다시 한번 삭제를 진행할 것인지 묻는다. confirmButtonText 옵션은 [확인] 버튼, cancelButtonText에는 [취소] 버튼에 표시될 텍스트를 입력할 수 있다. 사용자가 선택하면 프로미스 함수 then()이 실행된다.

15~23: 사용자가 삭제를 선택했다면, result.value 값이 true로 반환된다. Line 16에서 line 29의 <h1> 태그에 id 값(deleteId)으로 접근해 sweetalert2 영역을 화면에서 삭제한다. 그리고 삭제 완료 알림 창을 표시한다.

<실행 결과>

|  |
| --- |
| **<그림 77>** |

**<그림 78>**

059. fetch get 호출하기

**• 학습 내용:** fetch 함수로 get 방식 호출 방법을 이해한다.

**• 힌트 내용:** fetch 함수의 특징과 사용법에 대해 확인한다.

웹에서는 클라이언트와 서버가 http 프로토콜을 통해 요청과 응답을 주고받는다. http에서 사용하는 방식은 여러 가지가 있지만, GET과 POST를 가장 많이 사용한다. 간단히 정리하면 GET은 데이터를 조회해 가져와 사용하는 것이다.

|  |
| --- |
| NOTE |
| GET 방식은 <http://example.com?a=1&b=2>와 같이 url? 뒤에 파라미터명=값 형태로 필요한 데이터를 전달한다. 주로 데이터 조회나 검색 등의 기능에 사용된다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import FetchGet from './R059\_FetchGet'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <FetchGet/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R059\_FetchGet.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R059\_FetchGet.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R059\_FetchGet.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  class R059\_FetchGet extends Component {  componentDidMount = async () => {  const response = await fetch('http://date.jsontest.com/');  const body = await response.json();  alert(body.date)  }  render() {  return (  <h1>fetch get</h1>  )  }  }  export default R059\_FetchGet; |

자바스크립트 내장 함수인 fetch를 사용하면 쉽게 비동기 통신을 구현할 수 있다. 비동기 통신이란, 쉽게 말해 먼저 시작한 작업의 완료 여부와 상관없이 다음 작업을 실행하는 것이다.

5: fetch를 사용해서 get 방식으로 url을 호출해 데이터를 가져온다. 가져온 데이터를 response라는 변수에 할당한다. 이때 fetch 함수는 별도의 http 메서드를 설정이 없다면, get 방식으로 호출한다. 웹 브라우저에서 테스트 url을 호출하면, 다음과 같이 데이터를 확인할 수 있다(크롬 확장 프로그램 JSON Viewer 사용).

|  |
| --- |
|  |

**<그림 80>**

6: line 5에서 response라는 변수는 json 형태이기 때문에 json() 함수로 사용할 수 있도록 변환해 body라는 변수에 할당한다. 이때 fetch 함수의 비동기적 특징 때문에 line 5에서 데이터를 가져오기 전에 line 6이 실행돼 에러가 발생할 수 있다.

4~6: 이런 에러는 비동기 함수에 동기적인 기능을 추가해 해결할 수 있다. 이때 사용하는 것이 async와 await 문법이다. 비동기 함수를 실행하는 함수에 async를 붙이고 동기적으로 처리돼야 하는 함수 구문 앞에 await를 추가한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 79>**

60. fetch post 호출하기

**• 학습 내용:** fetch 함수로 post 방식 호출 방법을 이해한다.

**• 힌트 내용:** fetch 함수의 특징과 사용법에 대해 확인한다.

GET이 데이터를 조회해 가져와 사용하는 용도로 사용한다면, POST는 서버의 상태나 데이터를 변경하는 등의 수행 작업에 사용된다.

|  |
| --- |
| NOTE |
| POST 방식은 http body에 데이터를 넣어 전달하는데, url 뒤에 파라미터를 표시하지 않고 사용할 수 있다는 장점이 있다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import FetchPost from './R060\_FetchPost'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <FetchPost/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R060\_FetchPost.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R060\_FetchPost.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R060\_FetchPost.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | import React, { Component } from 'react';  class R060\_FetchPost extends Component {  componentDidMount = async () => {  const response = await fetch('http://date.jsontest.com/', {  method: 'POST',  headers: {  'Content-Type': 'application/json',  },  body: { a:"react", b:200 },  });  const body = await response.json();  alert(body.date)  }  render() {  return (  <h1>fetch post</h1>  )  }  }  export default R060\_FetchPost; |

get 방식과 마찬가지로 post도 자바스크립트 내장 함수인 fetch를 사용하면, 비동기 통신을 쉽게 구현할 수 있다.

5: fetch 사용 방법은 get과 비슷하지만, 두 번째 파라미터에 post 호출에 대한 정보가 추가된다.

6: post 방식으로 통신을 하겠다는 의미다.

7~9: http 통신을 할 때 header 부분에서 어떤 형태의 데이터를 사용할지 지정한다. 이때 Content-Type라는 변수를 사용한다. json 형태의 데이터를 사용하기 위해 application/json를 할당한다.

10: http body에 json 형태의 데이터를 담아 전송한다. Line 5에서 호출한 url에서는 body의 데이터를 사용하지 않기 때문에 어떤 형식으로 사용하는지만 확인한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 81>**

61. axios get 사용하기

**• 학습 내용:** axios 함수로 get 방식 호출 방법을 이해한다.

**• 힌트 내용:** axios 함수의 특징과 사용법에 대해 확인한다.

axios도 fetch와 마찬가지로 비동기 통신을 지원한다. axios는 fetch와 달리, 별도로 설치한 후 임포트해 사용해야 한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install –save axios]를 입력하면 다음과 같이 npm이 axios을 설치한다.

|  |
| --- |
|  |

**<그림 82>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import AxiosGet from './R061\_AxiosGet'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <AxiosGet/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R061\_AxiosGet.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R061\_AxiosGet.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R061\_AxiosGet.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  import axios from "axios";  class R061\_AxiosGet extends Component {  componentDidMount(){  axios.get('http://date.jsontest.com/')  .then( response => {alert(response.data.date)})  }  render() {  return (  <h1>axios get</h1>  )  }  }  export default R061\_AxiosGet; |

2: axios를 임포트해 사용할 수 있도록 한다.

6: axios.get(‘ 호출url’) 문법으로 get 방식의 http 호출을 한다.

7: line 6에서 url 호출이 완료되면, then 함수가 실행된다. 이때 호출 결과로 response가 반환된다. response와 호출된 변수명 사이에 data를 붙이면, 변수를 사용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 83>**

062. axios post 사용하기

**• 학습 내용:** axios 함수로 post 방식 호출 방법을 이해한다.

**• 힌트 내용:** axios 함수의 특징과 사용법에 대해 확인한다.

axios로 post 방식의 호출을 하는 문법은 get 방식으로 호출했을 때와 거의 동일하다. 다른 점은 post 함수의 파라미터로 json과 같은 형태의 데이터를 넣고 http body에 담아 url을 호출할 수 있다는 것이다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import AxiosPost from './R062\_AxiosPost'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <AxiosPost/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R062\_AxiosPost.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R062\_AxiosPost.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R062\_AxiosPost.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  import axios from "axios";  class R062\_AxiosPost extends Component {  componentDidMount(){  axios.post('http://date.jsontest.com/', {  a:"react", b:200  })  .then( response => {alert(response.data.date)})  }  render() {  return (  <h1>axios post</h1>  )  }  }  export default R062\_AxiosPost; |

2: axios를 임포트해 사용할 수 있도록 한다.

6~8: axios.post(‘ 호출 url’, json 데이터) 문법으로 post 방식의 http 호출을 한다. json 데이터는 {key1 : value1, key2 : value2} 형태로 사용하고 http body에 담겨 전송된다. Line 6에서 호출한 url에서는 body의 데이터를 사용하지 않기 때문에 어떤 형식으로 사용하는지만 확인한다.

9: line 6에서 url 호출이 완료되면, then 함수가 실행된다. 이때 호출 결과가 response가 반환된다. response와 호출된 변수명 사이에 data를 붙이면, 변수를 사용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 84>**

063. 콜백 함수 사용하기

**• 학습 내용:** 콜백 함수의 사용 방법을 이해한다.

**• 힌트 내용:** 콜백 함수의 특징과 단점을 확인한다.

자바스크립트는 비동기적으로 동작한다. 그래서 먼저 실행된 작업이 끝나지 않았더라도 다음 작업이 시작될 수 있다. 콜백 함수를 이용하면 특정 코드에 순서를 정해 원하는 시점에 실행할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import CallbackFunc from './R063\_CallbackFunc'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <CallbackFunc/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R063\_CallbackFunc.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R063\_CallbackFunc.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R063\_CallbackFunc.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import React, { Component } from 'react';  class R063\_CallbackFunc extends Component {  componentDidMount(){  this.logPrint(1, function(return1){  console.log("return1 : "+return1);  this.logPrint(return1, function(return2){  console.log("return2 : "+return2);  })  }.bind(this))  }  logPrint(param, callback) {  console.log("logPrint param : "+param);  param += param  callback(param);  }  render() {  return (  <h1>Callback Function</h1>  )  }  }  export default R063\_CallbackFunc; |

5: logPrint라는 함수를 호출하는데 파라미터로 1과 함수를 전달한다.

15~16: line 5에서 호출된 함수는 logPrint 함수는 전달받은 파라미터 param에 자기자신 1을 더해 2를 할당한다. 2가 된 param 변수를 callback 함수에 다시 파라미터로 넣고 함수를 실행한다.

5~10: line 16에서 실행된 callback 함수는 line 5의 function(return1){에서부터 line 10의 }.bind(this)까지다. Line 7에서 다시 한번 logPrint 함수를 실행하는데, line 5 첫 번째 함수 실행에서 반환된 return1 값(2)을 파라미터로 전달한다.

|  |
| --- |
| NOTE |
| line 7 콜백 함수 안에서 this는 line 5의 this와 다르다. 그래서 this로 logPrint 함수에 접근해 사용하려고 하면 에러가 발생한다. 함수 밖의 this를 함수 안에서도 동일하게 사용하기 위해서 .bind(this)를 함수에 붙여준다. |

콜백 함수를 사용해 원하는 순서대로 코드를 실행할 수 있었다. Line 5에서 1을 파라미터로 넣고 함수를 호출했고 2를 반환할 때까지 기다렸다가 line 7에서 다시 logPrint 함수를 호출했다. 하지만 콜백 함수가 증가할수록 함수 안에 또 다른 함수를 계속 추가해야 한다. 이런 형태를 콜백 지옥이라고 한다. 콜백 함수를 여러 번 사용할수록 코드가 더 지저분해진다는 단점이 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 85>**

064. Promise then 사용하기

**• 학습 내용:** Promise then 함수의 사용 방법을 이해한다.

**• 힌트 내용:** Promise then 함수의 특징과 콜백 함수와의 차이점을 확인한다.

promise는 콜백 함수와 같이 비동기적으로 동작하는 코드를 동기적으로 구현할 때 사용한다. promise를 사용하면 콜백 함수와 달리, 코드 가독성을 높일 수 있고 예외 처리도 쉽게 할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Promise from './R064\_Promise'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Promise/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R064\_Promise.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R064\_Promise.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R064\_Promise.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import React, { Component } from 'react';  class R064\_Promise extends Component {  componentDidMount(){  new Promise(resolve => {  setTimeout(function() {  resolve('react');  }, 1500);  })  .then(function(result) {  console.log(result);  return result + 200;  })  .then(result => {  console.log(result);  })  }  render() {  return (  <h1>Promise</h1>  )  }  }  export default R064\_Promise; |

promise에는 대기, 이행, 거부의 개념이 있다. 대기 상태에서 이행 상태로 변할 때 then() 함수 안의 코드가 실행된다.

5: Promise라는 객체를 생성해 promise를 사용한다. 파라미터로 하나의 함수(resolve)만 받을 경우, 이행 함수로 사용된다.

6~8: timeout 함수를 실행해 1.5초 후에 line 7의 resolve 함수가 실행된다. 이때 promise는 이행 상태가 되고 line 10의 then 함수가 실행된다.

10~13: line 7 resolve 함수에서 파라미터로 전달된 결괏값(react)을 result라는 파라미터로 받아 사용한다. result 변수에 200를 붙여 반환한다. 반환되는 순간 대기하고 있던 line 14의 then 함수가 실행된다.

14~16: line 12에서 반환된 값을 파라미터로 받아 사용한다. Line 10에서 사용한 function과 같은 기능을 하는 함수를 화살표 함수로 구현했다.

promise then 함수를 사용하면 콜백 함수의 계단식 복잡한 코드 형태보다 코드를 가독성 있게 구현할 수 있다.

<실행 결과>

|  |
| --- |
| **<그림 86>** |

065. Promise catch 사용하기

**• 학습 내용:** Promise catch 함수의 사용 방법을 이해한다.

**• 힌트 내용:** Promise catch 함수의 특징과 기능을 확인한다.

promise의 상태가 대기, 이행, 거부 중 거부 상태가 됐을 때 catch 함수를 실행한다. 대기 상태의 promise에 에러가 발생해 이행으로 상태 변화를 하지 못하는 경우다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Promise from './R065\_Promise'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Promise/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R065\_Promise.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R065\_Promise.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R065\_Promise.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  class R065\_Promise extends Component {  componentDidMount(){  new Promise((resolve, reject) => {  reject(Error("ERROR Info"));  })  .then(result => console.log("then "+result))  .catch(result => console.log("catch : "+result));  }  render() {  return (  <h1>Promise</h1>  )  }  }  export default R065\_Promise; |

5: Promise라는 객체를 생성해 promise를 사용한다. 파라미터로 2개의 함수를 받으면, 첫 번째 함수(resolve)는 이행 함수, 두 번째 함수(reject)는 거부 함수로 사용된다.

6: 거부 함수 reject를 실행하면서 Error 객체를 사용해 에러를 발생시킨다.

8: promise 동작 중 거부 함수가 실행되면, 이행 상태가 되지 못하기 때문에 then 함수는 실행되지 않는다.

9: promise의 상태가 거부로 변하면서 catch 함수가 실행된다. Line 6의 reject 함수에서 파라미터로 전달받은 에러 정보를 출력한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 87>**

(도비라)3부 중급: <저자 요청> 제목 요망(2부 도비라 참조)

**<저자 요청: 5~6줄 정도의 발문 부탁드립니다.>**

066. click 이벤트 사용하기(onClick)

**• 학습 내용:** onClick 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onClick으로 함수 호출 시 필요한 정보를 파라미터로 전달할 수 있다.

html과 달리 react에서 이벤트는 camelCase를 사용한다. onClick 이벤트는 특정 element가 클릭됐을 때 정의된 함수를 호출하는 방식으로 사용한다. html에서는 onclick으로 모두 소 문자로 나타낸다.

|  |
| --- |
| NOTE |
| camelCase란, 낙타의 등처럼 문자열 중간에 대 문자가 위로 올라오는 네이밍 문법이다. 여러 개의 단어가 합쳐질 때 단어가 시작되는 첫 글자만 대 문자로 표기하는 방식이다. 이때 맨 앞 글자가 소 문자면 lowerCamelCase, 대 문자면 UpperCamelCase라고 부른다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactonClick from './R066\_onClick'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactonClick/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R066\_onClick.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R066\_onClick.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R066\_onClick.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import React, { Component } from 'react';  class R065\_Promise extends Component {  buttonClick = (param) => {  if(typeof param != 'string') param = "Click a"  console.log('param : '+ param);  }  render() {  return (  <>  <button onClick={e => this.buttonClick("Click button")}>Click button</button>  <div onClick={e => this.buttonClick("Click div")}>Click div</div>  <a href="javascript:" onClick={this.buttonClick}>Click a</a>  </>  )  }  }  export default R065\_Promise; |

4: buttonClick이라는 함수를 선언하고 전달받은 파라미터를 param이라는 함수의 내부 변수에 넣어 사용한다.

5: param 변수가 문자열(string)이 아니라면, Click a라는 문자열을 param 변수에 할당한다.

6: param 변수를 로그로 출력한다.

12~14: <button>, <div>, <a> 태그에 onClick 이벤트를 추가한다. 사용자가 특정 태그 영역을 클릭할 때마다 buttonClick 함수를 호출하고 서로 다른 파라미터(Click button, Click div)를 전달한다. line 4 buttonClick 함수에서는 파라미터를 받는 것으로 선언돼 있지만, line 14처럼 파라미터 없이 함수를 호출해도 정상적으로 동작한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 88>**

067. change 이벤트 사용하기(onChange)

**• 학습 내용:** onChange 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onChange으로 함수 호출 시 필요한 정보를 이벤트 객체로 전달할 수 있다.

react에서 onChange 이벤트도 camelCase로 명칭을 사용한다. onChange 이벤트는 특정 element에 변화가 생겼을 때 정의된 함수를 호출하는 방식으로 사용한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactChange from './R067\_onChange'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactChange/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R067\_onChange.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R067\_onChange.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R067\_onChange.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  class R067\_onChange extends Component {  change = (e) => {  var val = e.target.value;  console.log('param : '+ val);  }  render() {  return (  <>  <input type="text" onChange={this.change}/>  <select onChange={this.change}>  <option value="react">react</option>  <option value="200">200</option>  </select>  </>  )  }  }  export default R067\_onChange; |

4: change이라는 함수를 선언하고 전달받은 파라미터를 e이라는 함수의 내부 변수에 넣어 사용한다.

5: 이벤트 객체 변수 e에 target 속성을 붙여 value 값을 가져오고 val 변수에 할당한다.

6: val 변수를 로그로 출력한다.

12: input 박스에 문자를 입력하거나 삭제할 때마다 onChange 이벤트가 발생해 change 함수를 호출한다. 함수 호출 시 따로 파라미터를 넘겨주지 않아도 이벤트 객체(line 4의 e 변수)가 전달된다 이벤트 객체에는 <input> 태그에 현재 입력된 값이 value로 할당된다.

13~16: select 박스에 선택된 option이 달라질 때마다 onChange 이벤트가 발생해 change 함수를 호출한다. 함수 호출 시 따로 파라미터를 넘겨주지 않아도 이벤트 객체(line 4의 e 변수)가 전달된다. 이벤트 객체에는 선택된 <option> 태그의 value 값이 할당된다.

실행 결과를 보면, input 박스에 글자 하나가 입력될 때마다 onchange 함수가 호출된 것을 확인할 수 있다. input 박스에 react라는 문자열을 입력한 후 select 박스에서 200이라는 option을 선택했을 때의 결과다.

<실행 결과>

|  |
| --- |
|  |

**<그림 89>**

068. mousemove 이벤트 사용하기(onMouseMove)

**• 학습 내용:** onMouseMove 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onMouseMove으로 함수 호출 시 필요한 정보를 파라미터로 전달할 수 있다.

react에서 onMouseMove 이벤트도 camelCase로 명칭을 사용한다. onMouseMove 이벤트는 특정 tag 영역 안에서 마우스 커서가 움직일 때 발생한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactMouseMove from './R068\_onMouseMove'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactMouseMove/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R068\_onMouseMove.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R068\_onMouseMove.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R068\_onMouseMove.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import React, { Component } from 'react';  class R068\_onMouseMove extends Component {  MouseMove(tag) {  console.log('TAG : '+tag);  }  render() {  return (  <>  <div onMouseMove={e => this.MouseMove("div")}>  <h3>DIV onMouseMove</h3>  </div>  <input type="text" onMouseMove={e => this.MouseMove("input")}/>  <select onMouseMove={e => this.MouseMove("select")}>  <option value="react">react</option>  <option value="200">200</option>  </select>  </>  )  }  }  export default R068\_onMouseMove; |

4: MouseMove이라는 함수를 선언하고 전달받은 파라미터를 tag라는 함수의 내부 변수에 넣어 사용한다.

6: tag 변수를 로그로 출력한다.

12~13: <div> 태그 영역에서 마우스 커서가 움직일 경우 MouseMove 함수를 호출한다. 파라미터로 넘긴 div이 line 5에서 출력된다.

14: <input> 태그 영역에서 마우스 커서가 움직일 경우 MouseMove 함수를 호출한다. 파라미터로 넘긴 input이 line 5에서 출력된다.

15~18: <select> 태그 영역에서 마우스 커서가 움직일 경우 MouseMove 함수를 호출한다. 파라미터로 넘긴 select이 line 5에서 출력된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 90>**

069. mouseover 이벤트 사용하기(onMouseOver)

**• 학습 내용:** onMouseOver 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onMouseOver으로 함수 호출 시 필요한 정보를 파라미터로 전달할 수 있다.

react에서 onMouseOver 이벤트도 camelCase로 명칭을 사용한다. onMouseOver 이벤트는 특정 tag 영역 안에 마우스 커서가 진입할 때 발생한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactMouseOver from './R069\_onMouseOver'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactMouseOver/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R069\_onMouseOver.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R069\_onMouseOver.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R069\_onMouseOver.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import React, { Component } from 'react';  class R069\_onMouseOver extends Component {  MouseOver(tag) {  console.log('TAG : '+tag);  }  render() {  return (  <>  <div onMouseOver={e => this.MouseOver("div")}>  <h3>DIV onMouseOver</h3>  </div>  <input type="text" onMouseOver={e => this.MouseOver("input")}/>  <select onMouseOver={e => this.MouseOver("select")}>  <option value="react">react</option>  <option value="200">200</option>  </select>  </>  )  }  }  export default R069\_onMouseOver; |

4: MouseOver이라는 함수를 선언하고 전달받은 파라미터를 tag라는 함수의 내부 변수에 넣어 사용한다.

6: tag 변수를 로그로 출력한다.

12~13: <div> 태그 영역에 마우스 커서가 진입할 경우 MouseOver 함수를 호출한다. 파라미터로 넘긴 div이 line 5에서 출력된다.

14: <input> 태그 영역에 마우스 커서가 진입할 경우 MouseOver 함수를 호출한다. 파라미터로 넘긴 input이 line 5에서 출력된다.

15~18: <select> 태그 영역에 마우스 커서가 진입할 경우 MouseOver 함수를 호출한다. 파라미터로 넘긴 select이 line 5에서 출력된다.

<실행 결과>

|  |
| --- |
|  |

**빈 박스 남겨주세요. 저자 그림 확인해야 함.**

070. mouseout 이벤트 사용하기(onMouseOut)

**• 학습 내용:** onMouseOut 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onMouseOut으로 함수 호출 시 필요한 정보를 파라미터로 전달할 수 있다.

react에서 onMouseOut 이벤트도 camelCase로 명칭을 사용한다. onMouseOut 이벤트는 특정 tag 영역 안에 마우스 커서가 진입한 후 벗어날 때 발생한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactMouseOut from './R070\_onMouseOut'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactMouseOut/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R070\_onMouseOut.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R070\_onMouseOut.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R070\_onMouseOut.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import React, { Component } from 'react';  class R070\_onMouseOut extends Component {  MouseOut(tag) {  console.log('TAG : '+tag);  }  render() {  return (  <>  <div onMouseOut={e => this.MouseOut("div")}>  <h3>DIV onMouseOut</h3>  </div>  <input type="text" onMouseOut={e => this.MouseOut("input")}/>  <select onMouseOut={e => this.MouseOut("select")}>  <option value="react">react</option>  <option value="200">200</option>  </select>  </>  )  }  }  export default R070\_onMouseOut; |

4: MouseOut이라는 함수를 선언하고 전달받은 파라미터를 tag라는 함수의 내부 변수에 넣어 사용한다.

5: tag 변수를 로그로 출력한다.

11~13: <div> 태그 영역에 마우스 커서가 진입한 후 벗어날 경우 MouseOut 함수를 호출한다. 파라미터로 넘긴 div이 line 5에서 출력된다.

14: <input> 태그 영역에 마우스 커서가 진입한 후 벗어날 경우 MouseOut 함수를 호출한다. 파라미터로 넘긴 input이 line 5에서 출력된다.

15~18: <select> 태그 영역에 마우스 커서가 진입한 후 벗어날 경우 MouseOut 함수를 호출한다. 파라미터로 넘긴 select이 line 5에서 출력된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 92>**

071. key 이벤트 사용하기(onKeyDown, onKeyPress, onKeyUp)

**• 학습 내용:** onKey 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onKeyDown, onKeyPress, onKeyUp 이벤트의 차이점을 확인한다.

react에서 onKeyDown, onKeyPress, onKeyUp 이벤트도 camelCase로 명칭을 사용한다. onKeyDown와 onKeyPress는 키가 눌렸을 때 동작한다. onKeyUp은 눌려 있던 키를 뗐을 때 동작한다.

|  |
| --- |
| NOTE |
| onKeyDown과 onKeyUp는 키가 눌리고 떼어지는 동작 자체에 반응한다. 문자, 숫자, [Ctrl], [Shift], [Alt], [F1]~[F12], [ScrLk], [Pause], [Enter] 를 인식하고 [한/영], [PrtSc] 키는 인식하지 못한다.  onKeyPress는 onKeyDown와 달리, 문자가 실제로 입력됐을 때 반응한다. [Ctrl], [Shift], [Alt], [F1]~[F12], [ScrLk], [Pause], [Enter], [한/영], [PrtSc] 키 등을 인식하지 못한다. |

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactOnKey from './R071\_OnKey'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactOnKey/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R071\_OnKey.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R071\_OnKey.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R071\_OnKey.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import React, { Component } from 'react';  class R071\_OnKey extends Component {  OnKey(event, e) {  var val = e.target.value;  console.log('event : '+event+", value: "+val);  }  render() {  return (  <>  onKeyDown : <input type="text" onKeyDown={e => this.OnKey("onKeyDown", e)}/><br/>  onKeyPress : <input type="text" onKeyPress={e => this.OnKey("onKeyPress", e)}/><br/>  onKeyUp : <input type="text" onKeyUp={e => this.OnKey("onKeyUp", e)}/>  </>  )  }  }  export default R071\_OnKey; |

4: OnKey이라는 함수를 선언하고 파라미터로 event, e 변수를 받는다. event는 문자열, e는 이벤트 객체 타입으로 전달받는다.

5: 이벤트 객체 변수 e에 target 속성을 붙여 value 값을 가져오고 val 변수에 할당한다.

6: event, val 변수를 로그로 출력한다.

12: onKeyDown input 박스에 [A], [Ctrl] 를 순서대로 입력하면, 실행 결과와 같이 [A]를 눌렀을 때 공백, [Ctrl]을 눌렀을 때 a가 출력된다. 첫 번째 출력에 공백이 출력되는 이유는 키가 눌렸을 때 onKeyDown 이벤트가 발생하고 <input> 태그의 value 값이 할당되기 때문이다.

13: onKeyPress input 박스에 [A], [Ctrl] 를 순서대로 입력하면, 실행 결과와 같이 [A]를 눌렀을 때 공백, [Ctrl]을 눌렀을 때 출력되지 않는다. 첫 번째 출력에 공백이 출력되는 이유는 키가 눌렸을 때 onKeyPress 이벤트가 발생하고 <input> 태그의 value 값이 할당되기 때문이다.

14: onKeyUp input 박스에 [A], [Ctrl] 를 순서대로 입력하면, 실행 결과와 같이 [A]를 눌렀을 때 공백, [Ctrl]을 눌렀을 때 a가 출력된다. 키가 눌렸을 때 <input> 태그에 입력된 a가 value 값에 할당되고 키가 떼어졌을 때 onKeyUp 이벤트가 발생하기 때문에 첫 번째 출력에서도 a가 출력된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 93>**

072. submit 이벤트 사용하기(onSubmit)

**• 학습 내용:** onSubmit 이벤트의 사용 방법을 이해한다.

**• 힌트 내용:** onSubmit 이벤트의 특징과 기능을 확인한다.

react에서 onSubmit 이벤트도 camelCase로 명칭을 사용한다. onSubmit 이벤트는 <form> 태그에 사용한다. <form> 태그 안에 있는 type이 submit인 <input> 태그를 클릭하거나 <input> 태그에 커서를 놓고 [Enter]를 누르면 onSubmit 이벤트가 발생한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactonSubmit from './R072\_onSubmit'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactonSubmit/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R072\_onSubmit.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R072\_onSubmit.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R072\_onSubmit.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import React, { Component } from 'react';  class R072\_onSubmit extends Component {  Submit(e) {  var inputValue = document.getElementById("inputId").value;  console.log("inputValue : "+inputValue);  e.preventDefault();  }  render() {  return (  <form onSubmit={this.Submit}>  <input type="text" name="inputName" id="inputId"/>  <input type="submit" value="Submit"/>  </form>  )  }  }  export default R072\_onSubmit; |

4~8: Submit이라는 함수를 선언하고 id 값이 inputId인 태그의 value 값을 inputValue 변수에 할당한다. inputValue를 로그로 출력한다. 이벤트 객체 e에 preventDefault 함수를 실행하면, submit이후에 페이지 새로 고침을 방지할 수 있다.

12~15: text 타입의 input 박스에 값을 입력하고 submit 타입의 버튼을 누르면 onSubmit 이벤트가 발생한다. 이때 Submit 함수가 실행돼 line 6에서 입력한 값이 출력된다. text 타입의 input 박스에 값을 입력하고 [Enter] 를 눌러도 동일하게 동작한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 94>**

073. Ref 사용하기

**• 학습 내용:** Ref의 사용 방법을 이해한다.

**• 힌트 내용:** Ref와 javascript가 react element에 접근하는 방식을 비교한다.

Ref는 reference의 약자로, 참조라는 뜻이다. react에서 element의 값을 얻거나 수정할 때 보통 javascript나 jquery를 사용한다. 이때 id나 class와 같은 속성으로 element에 접근한다. Ref를 사용하면 element가 참조하는 변수에 접근해 변경하고 element를 제어할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactRef from './R073\_ReactRef'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactRef/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R073\_ReactRef.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R073\_ReactRef.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R073\_ReactRef.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, { Component } from 'react';  class R073\_ReactRef extends Component {  constructor(props) {  super(props);  this.InputRef = React.createRef();  }  RefFocus = (e) => {  this.InputRef.current.focus();  }  JavascriptFocus() {  document.getElementById('id').focus();  }  render() {  return (  <>  <input id="id" type="text" ref={this.InputRef} />  <input type="button" value="Ref Focus" onClick={this.RefFocus}/>  <input type="button" value="Javascript Focus" onClick={this.JavascriptFocus}/>  </>  )  }  }  export default R073\_ReactRef; |

6: createRef 함수로 Ref 변수 inputRef를 생성한다.

20: element에 ref 속성을 추가하고 Ref 변수 inputRef를 할당해 참조하도록 한다. 이때 참조에 대한 정보가 ref의 current라는 속성에 할당된다.

9~11: line 21의 버튼을 클릭하면 RefFocus 함수가 실행된다. Line 20의 <input> 태그가 참조하고 있는 inputRef.current에 접근해 focus 이벤트를 발생시킨다.

13~15: line 22에서 버튼을 클릭하면 JavascriptFocus 함수가 실행된다. Javascript를 사용해 id 값으로 line 20 <input> 태그에 접근해 , focus 이벤트를 발생시킨다.

RefFocus 함수와 JavascriptFocus 함수는 동일하게 동작한다. input 박스에 focus 이벤트로 테두리를 굵게 처리한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 95>**

074. 커링 함수 구현하기

**• 학습 내용:** 커링의 사용 방법을 이해한다.

**• 힌트 내용:** 커링을 사용했을 때의 장점을 확인한다.

커링(Currying)은 함수의 재사용성을 높이기 위해 함수 자체를 return하는 함수다. 함수를 하나만 사용할 때는 필요한 모든 파라미터를 한 번에 넣어야 한다. 컬링을 사용하면 함수를 분리할 수 있으므로 파라미터도 나눠 전달할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import Currying from './R074\_ReactCurrying'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <Currying/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R074\_ReactCurrying.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R074\_ReactCurrying.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R074\_ReactCurrying.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | import React, { Component } from 'react';  class R074\_ReactCurrying extends Component {  plusNumOrString(c, d){  return c + d;  }  PlusFunc1(a){  return function(b){  return this.plusNumOrString(a, b)  }.bind(this)  }  PlusFunc2 = a => b => this.plusNumOrString(a, b)  PlusFunc(a){  return this.PlusFunc1(a)(200)  }  render() {  return (  <>  <input type="button" value="NumberPlus" onClick={e => alert(this.PlusFunc(100))}/>  <input type="button" value="StringPlus"  onClick={e => alert(this.PlusFunc("react"))}/>  </>  )  }  }  export default R074\_ReactCurrying; |

5~7: 파라미터로 전달된 c, d를 더해주는 함수를 생성한다. + 연산자는 숫자와 문자열에 사용할 수 있다.

9~13: PlusFunc1 함수는 this.plusNumOrString(a, b)을 return하는 또 다른 함수를 return한다.

15: PlusFunc2 함수는 PlusFunc1 함수를 화살표 함수로 간단하게 표현한 것이다. 완전히 동일한 함수다.

17~19: PlusFunc 함수는 파라미터를 1개(a)만 받는다. PlusFunc1 함수를 호출할 때 변수 a와 상수 200을 함께 파라미터로 전달한다.

24~25: PlusFunc 함수를 호출하면서 숫자 100과 문자열 react를 전달한다. 고정으로 더해주는 숫자 200은 line 18에서 파라미터로 넣어주기 때문에 PlusFunc 함수를 호출할 때 각각 200을 전달하지 않아도 된다.

NumberPlus 버튼 클릭 시 100+200의 결과인 300이 StringPlus 버튼 클릭 시 react+200의 결과인 react200이 팝업 창에 표시된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 96>**

075. 하이오더 컴포넌트 구현하기

**• 학습 내용:** 하이오더 컴포넌트의 사용 방법을 이해한다.

**• 힌트 내용:** 하이오더 컴포넌트와 커링을 비교해 특징을 확인한다.

커링과 같이 함수 자체를 인자로 받거나 반환하는 함수를 고차 함수라고 한다. 이와 비슷하게 컴포넌트를 인자로 받거나 반환하는 함수를 고차 컴포넌트(HOC, Higher-Order Component)라고 한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ReactHoc from './Hoc/R075\_ReactHoc'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ReactHoc name='React200'/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 Hoc 폴더의 R075\_ReactHoc.js 파일을 임포트해 사용할 수 있도록 한다.

8: R075\_ReactHoc 컴포넌트에 name 값을 React200으로 할당해 전달한다.

src 폴더에 Hoc 폴더를 만들고 R075\_ReactHoc.js 파일을 생성해 다음과 같이 입력한다.

<소스> R075\_ReactHoc.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12 | import React from 'react'  import withHocComponent from "./withHocComponent";  class R075\_ReactHoc extends React.Component {  render () {  console.log('2. HocComponent render')  return (  <h2>props.name : {this.props.name}</h2>  )  }  }  export default withHocComponent(R075\_ReactHoc, 'R075\_ReactHoc') |

2: 같은 폴더 경로에 있는 withHocComponent 컴포넌트를 사용할 수 있도록 임포트한다.

6~9: render 함수가 실행되면, 로그를 출력하고 props 데이터에 있는 name 변수를 화면에 그려준다.

12: withHocComponent 컴포넌트를 호출하면서 R075\_ReactHoc 컴포넌트와 컴포넌트명을 파라미터로 넘긴다. 이때 R075\_ReactHoc 컴포넌트는 익스포트(export)되지 않기 때문에 render 함수가 실행되지 않는다.

Hoc 폴더에 withHocComponent.js 파일을 생성해 다음과 같이 입력한다.

<소스> withHocComponent.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12 | import React from "react";  export default function withHocComponent(InComponent, InComponentName) {  return class OutComponent extends React.Component {  componentDidMount () {  console.log(`3. InComponentName : ${InComponentName}`)  }  render () {  console.log('1. InComponent render')  return (<InComponent {...this.props}/>)  }  }  } |

2: R075\_ReactHoc 컴포넌트의 line 12에서 withHocComponent 컴포넌트를 익스포트하면서 전달한 파라미터를 받는다.

4~6: render 함수가 실행된 후 파라미터로 전달받은 컴포넌트명 InComponentName 변수를 로그로 출력한다.

9: 파라미터로 전달받은 InComponent 변수는 R075\_ReactHoc 컴포넌트 자체다. R075\_ReactHoc 컴포넌트를 return하면서 props 값을 전달한다. props에는 App.jsp에서 전달한 name 변수가 있다. 컴포넌트가 return되면 R075\_ReactHoc 컴포넌트의 render 함수가 실행되고 props.name 값이 화면에 출력된다.

하이오더 컴포넌트를 구현하면, 여러 컴포넌트에 동일하게 적용되야 하는 공통 기능을 코드 중복 없이 사용할 수 있다. withHocComponent.js 코드에서 예를 들면, line 8의 console.log 함수를 모든 컴포넌트에서 출력해야 하는데, hoc를 구현하지 않았다면 각각의 컴포넌트에서 동일한 코드를 작성해야 한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 97>**

076. 컨텍스트 API 사용하기

**• 학습 내용:** 컨텍스트 API의 사용 방법을 이해한다.

**• 힌트 내용:** 컨텍스트 API와 props를 비교해 특징을 확인한다.

props를 사용하면 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전송할 수 있다. 그런데 만약 손자 컴포넌트가 부모 컴포넌트의 데이터를 필요하다면, 자식 컴포넌트가 중간에서 데이터를 전달해야 한다. 이때 자식 컴포넌트가 부모 컴포넌트의 데이터가 필요하지 않은 상황이라면 불필요한 코드를 작성하게 된다. 컨텍스트는 데이터의 공급자와 소비자를 정의하고 데이터가 필요한 컴포넌트만 사용할 수 있게 구현할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ContextApi from './Context/R076\_ContextApi'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ContextApi/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 Context 폴더의 R076\_ContextApi.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 Context 폴더를 만들고 R076\_ContextApi.js 파일을 생성해 다음과 같이 입력한다.

<소스> R076\_ContextApi.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16 | import React from 'react'  import Children from "./contextChildren";  const {Provider, Consumer} = React.createContext()  export {Consumer}  class R076\_ContextApi extends React.Component {  render () {  return (  <Provider value='React200'>  <Children />  </Provider>  )  }  }  export default R076\_ContextApi |

2: 같은 폴더 경로에 있는 contextChildren 컴포넌트를 사용할 수 있도록 임포트한다.

4: 리액트 기본 제공 함수인 createContext를 호출하고 공급자 Provider와 소비자 Consumer를 받아 사용할 수 있도록 한다.

5: 하위 컴포넌트에서 소비자를 사용할 수 있도록 익스포트한다.

10~12: 자식 컴포넌트를 <Provider> 태그로 감싸고 전달할 데이터를 value 값으로 할당한다.

Context 폴더에 contextChildren.js 파일을 생성해 다음과 같이 입력한다.

<소스> contextChildren.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11 | import React from 'react'  import Children2 from "./contextChildren2";  class contextChildren extends React.Component {  render () {  return (  <Children2 />  )  }  }  export default contextChildren |

7: contextChildren 컴포넌트에서는 부모 컴포넌트의 데이터를 사용하지 않고 손자 컴포넌트 contextChildren2를 return한다. 손자 컴포넌트에서 사용할 데이터를 전달하지 않아도 된다.

Context 폴더에 contextChildren2.js 파일을 생성해 다음과 같이 입력한다.

<소스> contextChildren2.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13 | import React from 'react'  import {Consumer} from "./R076\_ContextApi";  class contextChildren2 extends React.Component {  render () {  return (  <Consumer>  {contextValue=> <h3>{`contextValue : ${contextValue}`}</h3>}  </Consumer>  )  }  }  export default contextChildren2 |

2: 부모 컴포넌트의 데이터를 사용하기 위해 R076\_ContextApi 컴포넌트에서 익스포트했던 Consumer를 임포트해 사용할 수 있게 한다.

7~9: <Consumer> 태그로 출력할 element를 감싸고 R076\_ContextApi 컴포넌트에서 value에 할당했던 데이터를 contextValue 변수로 받아 출력한다.

컨텍스트를 사용하면 하위 컴포넌트가 여러 개인 구조에서 유용하게 사용할 수 있다. 몇 번째 하위 컴포넌트인지 관계 없이 필요한 하위 컴포넌트에서 소비자를 임포트해 필요한 데이터를 사용할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 98>**

077. 컨텍스트로 부모 데이터 변경하기

**• 학습 내용:** 컨텍스트로 부모 컴포넌트에 접근하는 방법을 이해한다.

**• 힌트 내용:** 컨텍스트 API와 props를 비교해 특징을 확인한다.

Props는 데이터가 부모에서 자식 컴포넌트로 단방향으로만 이동할 수 있다. 컨텍스트를 사용하면 자식 컴포넌트에서 부모 컴포넌트의 데이터를 변경할 수 있다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import ContextApi from './Context/R077\_ContextApi'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <ContextApi/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 Context 폴더의 R077\_ContextApi.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 Context 폴더를 만들고 R077\_ContextApi.js 파일을 생성해 다음과 같이 입력한다.

<소스> R077\_ContextApi.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React from 'react'  import Children from "./contextChildren";  const {Provider, Consumer} = React.createContext()  export {Consumer}  class R077\_ContextApi extends React.Component {  constructor (props) {  super(props);  this.setStateFunc = this.setStateFunc.bind(this)  }  setStateFunc(value) {  this.setState({name : value});  }  render () {  const content = {  ...this.state,  setStateFunc: this.setStateFunc  }  return (  <Provider value={content}>  <Children />  </Provider>  )  }  }  export default R077\_ContextApi |

12~14: state 변수 name에 파라미터 value를 할당하는 함수를 선언한다.

17~20: content 변수에 R077\_ContextApi 컴포넌트의 state와 setStateFunc 함수를 할당한다.

22~24: 자식 컴포넌트를 <Provider> 태그로 감싸고 전달할 데이터인 content를 value 값으로 할당한다.

Context 폴더에 contextChildren.js 파일을 생성해 다음과 같이 입력한다.

<소스> contextChildren.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17 | import React from 'react'  import {Consumer} from "./R077\_ContextApi";  class contextChildren extends React.Component {  render () {  return (  <Consumer>  {contextValue=>  <button onClick={e => contextValue.setStateFunc("react200")}>  {contextValue.name}\_button  </button>  }  </Consumer>  )  }  }  export default contextChildren |

7~13: <Consumer> 태그로 출력할 element를 감싸고 R077\_ContextApi 컴포넌트에서 value에 할당했던 데이터를 contextValue 변수로 받아 사용한다.

9: 버튼을 클릭하면, 파라미터로 전달받은 R077\_ContextApi 컴포넌트의 setStateFunc("react200")을 호출한다. 이때 R077\_ContextApi 컴포넌트의 state 변수 name 값을 react200으로 할당한다.

10: 버튼을 누르기 전 contextValue.name 값이 없기 때문에 버튼명이 \_button으로 표시된다. 버튼을 누르면 변경된 R077\_ContextApi 컴포넌트의 state 변수 name 값인 react200을 가져오고 버튼명이 react200\_button으로 표시된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 99>**

078. redux 리듀서로 스토어 생성

**• 학습 내용:** redux에서 리듀서로 스토어를 생성하는 방법을 이해한다.

**• 힌트 내용:** redux에서 스토어와 리듀서의 역할을 확인한다.

redux는 컨텍스트와 마찬가지로 데이터를 필요한 컴포넌트에서만 요청해 사용할 수 있다. 컨텍스트는 부모 컴포넌트에서 생성한 데이터에 모든 자식 컴포넌트에서 접근할 수 있다. 하지만 redux에서는 컴포넌트 외부의 스토어라는 곳에서 관리한다. 그래서 컴포넌트의 위치에 상관없이 스토어에 접근해 데이터를 사용하고 변경할 수 있다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install --save redux]를 입력하면 다음과 같이 npm이 redux을 설치한다.

|  |
| --- |
|  |

**<그림 100>**

redux는 스토어 > 컴포넌트 > 액션 > 리듀서 > 다시 스토어의 과정을 통해 데이터를 변경한다.

78~80번 예제에서 사용할 파일 구조는 다음과 같다. 예제 내용은 버튼을 눌렀을 때 초깃값 react글자에 문자열 200을 붙여주는 것이다.

|  |  |  |
| --- | --- | --- |
| src | actions | index.js |
|  | reducers | index.js |
|  | Index.js |  |
|  | App.js |  |
|  | StrAddButton.js |  |

src 폴더의 index.js 파일을 다음과 같이 수정한다.

<소스> index.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import {createStore} from 'redux';  import reducers from './reducers';  const store = createStore(reducers);  const listener = ()=> {  ReactDOM.render(  <App store={store}/>,  document.getElementById('root')  );  };  store.subscribe(listener);  listener(); |

5: redux 패키지에서 스토어 생성 함수 createStore를 임포트해 사용할 수 있도록 한다.

8: createStore 함수의 파라미터로 reducers 폴더 경로를 넘긴다. reducers 폴더의 Index.js에는 데이터 초깃값을 설정하고 데이터를 변경해주는 함수가 있다. 이 함수를 ‘리듀서’라고 한다.

10~15: line 8에서 생성한 store를 App 컴포넌트에 전달한다. render 함수를 listener라는 함수 내부에 위치 시킨다.

17: store를 구독하면 store 데이터에 변화가 있을 때 listener 함수 내부의 render 함수를 실행하고 변경된 데이터를 렌더링한다.

18: line 10에서 render 함수를 listener 함수로 감쌌기 때문에 초기 렌더링을 위해 수동으로 render 함수를 실행시켜준다.

78~80번의 실행 결과는 80번에서 확인한다.

79. redux 스토어 상속과 디스패치 함수 실행

**• 학습 내용:** redux에서 스토어 상속과 디스패치 함수를 이해한다.

**• 힌트 내용:** redux에서 스토어를 props에 담아 넘길 수 있다.

props에 스토어를 담아 하위 컴포넌트로 전달하면, 전달받은 컴포넌트에서 스토어에 접근할 수 있다. 컴포넌트에서 dispatch 함수를 사용하면 스토어 데이터를 변경할 수 있다.

src 폴더의 App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React, {Component} from 'react'  import StrAddButton from './StrAddButton'  class App extends Component {  render() {  return (  <div>  <h1>Start React 200!</h1>  <span>{this.props.store.getState().data.str}</span><br/>  <StrAddButton store={this.props.store}/>  </div>  );  }  }  export default App; |

2: 버튼 컴포넌트 StrAddButton를 임포트해 사용할 수 있도록 한다.

9: index.js에서 전달받은 store에 props를 통해 접근한다. 스토어 state 데이터에서 str 변숫값을 가져온다.

10: 버튼 컴포넌트 StrAddButton에 store를 전달한다.

src 폴더의 StrAddButton.js 파일을 다음과 같이 수정한다.

<소스> StrAddButton.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import React, {Component} from 'react';  import {add} from './actions'  class StrAddButton extends Component {  render() {  return (  <input value='Add200' type="button" onClick={this.addString}/>  )  }  addString = () => {  this.props.store.dispatch(add());  }  }  export default StrAddButton; |

2: actions 폴더 경로를 임포트한다. actions 폴더의 index.js 파일에는 add라는 함수가 있다.

7: 버튼을 클릭하면 addString 함수를 실행한다.

12: dispatch 함수를 통해 add 함수(actions 폴더의 index.js 파일)의 반환 값을 스토어에 전달한다. 이때 add 함수의 type 값을 리듀서(reducers 폴더의 Index.js)에서 참고해 스토어 데이터를 변경한다. 결론적으로 dispatch 함수는 리듀서에 액션을 전달하고 리듀서에서는 액션에 작성된 작업 내용을 읽어 스토어 데이터를 변경한다.

78~80번의 실행 결과는 80번에서 확인한다.

080. redux 리듀서에서 스토어 데이터 변경

**• 학습 내용:** redux에서 액션과 리듀서 함수를 이해한다.

**• 힌트 내용:** redux에서 액션에는 리듀서가 처리할 작업 정보가 있다.

컴포넌트에서 dispatch 함수가 실행되면, 리듀서 함수는 액션 값을 참조해 작업을 실행한다. 이때 액션 값에 따라 조건을 분기할 수 있다.

actions 폴더의 index.js 파일을 다음과 같이 수정한다.

<소스> index.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6 | export const ADD = 'ADD';  export const add = () => {  return {  type: ADD  }  }; |

2~6: StrAddButton 컴포넌트에서 add 함수를 실행하면 type 변수에 ADD 변숫값을 할당해 return한다.

reducers 폴더의 index.js 파일을 다음과 같이 수정한다.

<소스> StrAddButton.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | import {ADD} from '../actions'  import {combineReducers} from 'redux'  const initState = {  str: 'react',  };  const data = (state = initState, action) => {  switch (action.type) {  case ADD:  return state, {  str: state.str + '200'  };  default:  return state;  }  };  const App = combineReducers({  data  });  export default App; |

1: actions 폴더 경로를 임포트한다. actions 폴더의 index.js에서 ADD 변숫값을 가져온다.

2: 리듀서를 스토어에 넘겨주기 위해 combineReducers 함수를 임포트한다.

4~6: 리듀서 데이터의 초깃값을 선언, 할당한다.

8: state 변수에 line 5에서 할당한 초깃값이 할당된다

9: add 함수에서 반환된 type 변숫값 ADD를 가져온다.

10~13: action.type 값이 ADD값과 같은 경우, state 변수 str에 200을 붙인다. 반환 값은 line 8의 data 변수에 할당된다.

14~16: action.type 값이 ADD값과 같지 않은 경우, state 변수를 그대로 반환한다.

19~21: 리듀서 함수 data를 combineReducers 함수를 이용해 하나의 리듀싱 함수로 변환하고 익스포트한다. 이 함수는 src의 index.js에 있는 createStore 함수의 파라미터로 넘겨진다. 스토어 state 값에 변경이 발생했기 때문에 subscribe 함수가 동작해 화면이 렌더링된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 101>**

081. react-redux로 스토어 상속

**• 학습 내용:** react-redux에서 스토어를 상속 방법을 이해한다.

**• 힌트 내용:** redux와 비교해서 react-redux의 장점을 확인한다.

redux만 사용해도 충분히 스토어 데이터를 사용하고 변경할 수 있다. react-redux는 redux를 react와 연동해서 사용하기 편리하도록 만든 라이브러리다. react-redux의 장점은 크게 두 가지가 있다. 첫 번째는 store를 하위 컴포넌트에 매번 상속하지 않고 사용할 수 있다. 두 번째는 스토어 데이터를 사용, 변경하는 코드를 모듈화해 컴포넌트 내에 중복된 코드 사용을 최소화할 수 있다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install --save react-redux]를 입력하면 다음과 같이 npm이 react-redux를 설치한다.

|  |
| --- |
|  |

**<그림 102>**

react-redux도 redux와 마찬가지로 스토어 > 컴포넌트 > 액션 > 리듀서 > 다시 스토어의 과정을 통해 데이터를 변경한다. 차이점은 스토어 > 컴포넌트, 컴포넌트 > 액션 단계에서 connect라는 react-redux 패키지 함수가 사용된다.

81~83번 예제는 78~80번 예제와 동일하다. react-redux를 사용하면서 추가되거나 변경된 코드만 확인한다. actions 폴더의 index.js와 reducers 폴더의 index.js 파일은 변경사항이 없다.

|  |  |  |
| --- | --- | --- |
| src | actions | index.js |
|  | reducers | index.js |
|  | Index.js |  |
|  | App.js |  |
|  | StrAddButton.js |  |

src 폴더의 index.js 파일을 다음과 같이 수정한다.

<소스> index.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import {createStore} from 'redux';  import {Provider} from 'react-redux'  import reducers from './reducers';  const store = createStore(reducers);  const listener = ()=> {  ReactDOM.render(  <Provider store={store}>  <App indexProp="react"/>  </Provider>,  document.getElementById('root')  );  };  store.subscribe(listener);  listener(); |

9: store 상속을 위해 react-redux의 Provider를 임포트해 사용할 수 있도록 한다.

13~15: <Provider> 태그로 App 컴포넌트를 감싸는 부분이 변경됐다. Provider에 데이터를 넘겨주면 중간 컴포넌트에서 props 값을 다시 전달해줄 필요 없이 모든 하위 컴포넌트에서 데이터 사용이 가능하다. 컨텍스트 API에서 사용했던 Provider와 동일한 기능을 한다.

14: App 컴포넌트에서 사용할 변수 indexProp에 react 문자열을 할당해 props로 전달한다.

81~83번의 실행 결과는 83번에서 확인한다.

082. react-redux로 스토어 데이터 사용

**• 학습 내용:** react-redux로 스토어 데이터를 사용하는 방법을 이해한다.

**• 힌트 내용:** react-redux 패키지의 connect 함수 사용 형태를 확인한다.

react-redux 패키지의 connect 함수는 파라미터를 4개까지 받을 수 있는데, 파라미터 위치에 따라 미리 정의된 함수나 object를 사용할 수 있다. 예제에서는 2개의 파라미터를 사용한다. 첫 번째 위치의 파라미터(mapStateToProps)는 스토어의 상태 값을 컴포넌트 props에 할당하는 함수다. 두 번째 파라미터(mapDispatchToProps)는 dispatch 함수를 컴포넌트 함수에 바인딩하는 함수다.

src 폴더의 App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, {Component} from 'react'  import {connect} from 'react-redux'  import StrAddButton from './StrAddButton'  class App extends Component {  render() {  return (  <div>  <h1>Start React 200!</h1>  {/\* <span>{this.props.store.getState().data.str}</span><br/> \*/}  <span>{this.props.str}</span><br/>  {/\* <StrAddButton store={this.props.store}/> \*/}  <StrAddButton AppProp="200"/>  </div>  );  }  }  let mapStateToProps = (state, props) => {  console.log('Props from indes.js : ' + props.indexProp)  return {  str: state.data.str,  };  };  App = connect(mapStateToProps, null)(App);  export default App; |

2: react-redux 패키지에서 connect 함수를 임포트해 사용할 수 있도록 한다.

26: connect 함수의 첫 번째 파라미터는 mapStateToProps 함수로, 스토어의 state 값에 접근할 수 있다.

19: mapStateToProps 함수는 첫 번째 파라미터로 스토어의 state 변수를 두 번째 파라미터로 부모 컴포넌트에서 전달한 props 변수를 받는다.

20: index.js에서 전달한 props 변수 indexProp를 콘솔에 출력한다.

21~23: 스토어의 state 변수 str값을 App 컴포넌트 props의 str 변수로 할당한다.

11: line 22에서 str 변수로 할당한 값을 화면에 출력한다. redux를 사용하던 line 10과 비교하면 컴포넌트 내에 코드가 짧아진 것을 확인할 수 있다. 접근하는 스토어 변수가 많아질수록 코드 효율이 좀 더 좋아지게 된다.

13: line 12 기존 코드에서는 하위 버튼 컴포넌트에 store를 props로 다시 전달했지만, index.js에서 Provider를 사용했기 때문에 전달하지 않아도 된다. StrAddButton 컴포넌트에서 사용할 변수 AppProp에 200 문자열을 할당해 props로 전달한다.

81~83번의 실행 결과는 83번에서 확인한다.

083. react-redux로 스토어 데이터 변경

**• 학습 내용:** react-redux로 스토어 데이터를 변경하는 방법을 이해한다.

**• 힌트 내용:** react-redux 패키지의 connect 함수 사용 형태를 확인한다.

connect 함수의 두 번째 파라미터 mapDispatchToProps 함수로, dispatch 함수를 컴포넌트 함수에 바인딩할 수 있다. 즉, 컴포넌트 함수가 실행되면 바인딩된 dispatch 함수가 실행된다.

src 폴더의 StrAddButton.js 파일을 다음과 같이 수정한다.

<소스> StrAddButton.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import React, {Component} from 'react';  import {connect} from 'react-redux';  import {add} from './actions'  class StrAddButton extends Component {  render() {  return (  // <input value='Add200' type="button" onClick={this.addString}/>  <input value='Add200' type="button" onClick={this.props.addString}/>  )  }  // addString = () => {  // this.props.store.dispatch(add());  // }  }  let mapDispatchToProps = (dispatch, props) => {  console.log('Props from App.js : ' + props.AppProp)  return {  addString: () => dispatch(add())  };  };  StrAddButton = connect(null, mapDispatchToProps)(StrAddButton);  export default StrAddButton; |

2: react-redux 패키지에서 connect 함수를 임포트해 사용할 수 있도록 한다.

25: connect 함수의 두 번째 파라미터는 mapDispatchToProps 함수로, 리듀서에 액션을 전달하는 함수인 dispatch를 인자로 받아 사용할 수 있다.

18: mapDispatchToProps 함수는 첫 번째 파라미터로 dispatch 함수를 두 번째 파라미터로 부모 컴포넌트에서 전달한 props 변수를 받는다.

19: App.js에서 전달한 props 변수 AppProp를 콘솔에 출력한다.

20~22: dispatch 함수를 컴포넌트 내 함수인 addString에 바인딩한다. 이때 addString 함수는 props에 할당된다.

13~15: 기존의 addString 함수는 dispatch 함수를 호출하기 위해 컴포넌트 내에 새로운 함수를 선언했다. line 21에서 두 함수를 바인딩했기 때문에 별도의 함수 선언은 불필요하다.

9: 버튼을 클릭하면 props에 할당된 addString 함수를 실행한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 103>**

084. redux 미들웨어 사용하기

**• 학습 내용:** redux로 미들웨어를 사용하는 방법을 이해한다.

**• 힌트 내용:** redux 패키지의 applyMiddleware 함수 사용 형태를 확인한다.

redux 미들웨어는 액션을 dispatch 함수로 전달하고 리듀서가 실행되기 전과 리듀서 실행 후에 처리되는 기능을 말한다. redux 패키지에서 지원하는 applyMiddleware 함수를 사용하면 간단하게 미들웨어를 구현할 수 있다.

81~83번 예제에 사용한 코드에서 index.js 파일만 수정해 미들웨어 동작을 확인한다.

src 폴더의 index.js 파일을 다음과 같이 수정한다.

<소스> index.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import App from './App';  import {createStore, applyMiddleware} from 'redux';  import {Provider} from 'react-redux'  import reducers from './reducers';  const CallMiddleware = store => nextMiddle => action => {  console.log('1. reducer 실행 전');  console.log('2. action.type : '+action.type+', store str : '+store.getState().data.str);  let result = nextMiddle(action);  console.log('3. reducer 실행 후');  console.log('4. action.type : '+action.type+', store str : '+store.getState().data.str);  return result;  }  const store = createStore(reducers, applyMiddleware(CallMiddleware));  const listener = ()=> {  ReactDOM.render(  <Provider store={store}>  <App indexProp="react"/>  </Provider>,  document.getElementById('root')  );  };  store.subscribe(listener);  listener(); |

5: redux 패키지에서 applyMiddleware 함수를 임포트해 사용할 수 있도록 한다.

18: createStore 함수의 두 번째 파라미터를 applyMiddleware 함수로 전달한다. applyMiddleware 함수의 파라미터로는 새로 정의해 사용할 미들웨어 함수명을 넣는다.

9: applyMiddleware 함수에 전달될 함수는 다중 컬링 구조로, 세 가지 인자를 순서대로 받는다. 첫 번째 인자는 스토어, 두 번째 인자는 다음 미들웨어를 호출하는 함수로 예제에서는 미들웨어가 1개이기 때문에 리듀서를 호출한다. 세 번째 인자는 액션이다.

11: 리듀서 실행 전 액션의 type 변수와 store의 상태 변수 str의 값을 로그로 출력한다.

12: 다음 실행할 미들웨어가 없으므로 리듀서를 실행한다.

14: 리듀서를 실행하기 전 액션의 type 변수와 store의 상태 변수 str의 값을 로그로 출력한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 104>**

085. react-cookies save 사용하기

**• 학습 내용:** react-cookies의 save 함수 사용 방법을 이해한다.

**• 힌트 내용:** save 함수 호출 시 웹 브라우저에 쿠키 값을 저장할 수 있다.

쿠키란, 사용자가 접속한 웹 사이트의 서버를 통해 사용자 컴퓨터에 설치되는 정보를 말한다. 보통 크롬과 같은 웹 브라우저에 쿠키가 저장된다. 쿠키는 사용자 정보를 저장하거나 마케팅을 위한 목적으로 사용될 수 있다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install react-cookies --save]를 입력하면 다음과 같이 npm이 react-cookies를 설치한다.

|  |
| --- |
|  |

**<그림 105>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import CookieSave from './R085\_cookieSave'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <CookieSave/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R085\_cookieSave.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R085\_cookieSave.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R085\_cookieSave.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import React, { Component } from 'react';  import cookie from 'react-cookies';  class R085\_cookieSave extends Component {  componentDidMount() {  const expires = new Date()  expires.setMinutes(expires.getMinutes() + 60)  cookie.save('userid', "react200"  , {  path: '/',  expires,  // secure: true,  // httpOnly: true  }  );  }  render() {  return (  <><h3>react-cookies Save</h3></>  )  }  }  export default R085\_cookieSave; |

2: react-cookies 패키지를 임포트해 사용할 수 있도록 한다.

6: 현재 날짜와 시간을 불러와 expires 변수에 할당한다.

7: getMinutes 함수를 사용해 expires 변수를 분으로 환산하고 60분을 더한 값을 setMinutes 함수로다시 할당한다.

8: react-cookies 패키지 save 함수는 세 가지 파라미터를 받는다. 첫 번째는 쿠키의 키(userid), 두 번째는 쿠키 값(react200), 세 번째는 쿠키 옵션이다.

9~14: 세 번째 인자인 쿠키 옵션을 할당하는 부분이다.

10: path는 쿠키 값을 저장하는 서버 경로다. 예를 들어, path가 /react라면 localhost:3000/에서는 저장된 쿠를 확인할 수 없고 localhost:3000/react에서만 확인할 수 있다. path가 /라면 모든 페이지에서 쿠키에 접근할 수 있다.

11: 옵션의 두 번째 인자는 쿠키 유효 시간이다. Line 7에서 60분을 더했기 때문에 쿠키가 저장된 이후 1시간 동안 웹 브라우저에 쿠키가 남아 있게 된다.

12~13: secure가 true면 웹 브라우저와 웹 서버가 https로 통신하는 경우에만 쿠키가 저장된다. httpOnly 옵션은 document.cookie라는 자바스크립트 코드로 쿠키에 비정상적으로 접속하는 것을 막는 옵션이다. secure와 httpOnly 옵션은 예제에서 사용하지 않는다.

크롬 개발자 도구 > Application > Cookies > <http://localhost:300>을 클릭하면 다음과 같이 저장된 쿠키 값과 만료시간을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 106>**

086. react-cookies load 사용하기

**• 학습 내용:** react-cookies의 load 함수 사용 방법을 이해한다.

**• 힌트 내용:** load 함수 호출 시 웹 브라우저에서 쿠키 값을 불러올 수 있다.

save 함수가 쿠를 (key, value) 쌍으로 저장하는 함수였다면, load 함수는 웹 브라우저에 남아 있는 쿠키에 key로 접근해 value을 가져오는 함수다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import CookieLoad from './R086\_cookieLoad'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <CookieLoad/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R086\_cookieLoad.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R086\_cookieLoad.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R086\_cookieLoad.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, { Component } from 'react';  import cookie from 'react-cookies';  class R086\_cookieLoad extends Component {  componentDidMount() {  const expires = new Date()  expires.setMinutes(expires.getMinutes() + 60)  cookie.save('userid', "react200"  , {  path: '/',  expires,  // secure: true,  // httpOnly: true  }  );  setTimeout(function() {  alert(cookie.load('userid'))  },1000);  }  render() {  return (  <><h3>react-cookies Load</h3></>  )  }  }  export default R086\_cookieLoad; |

16~18: setTimeout 함수를 사용해 1초 후에 alert을 띄우도록 한다. 쿠키가 저장되기까지 충분한 시간을 두기 위해 사용한다.

17: cookie.load(key) 형태로 웹 브라우저에 저장된 쿠키 중 userid라는 key를 가진 쿠키의 value를 가져온다. Line 8에서 저장된 쿠를 1초 후 가져와 alert 함수로 값을 표시한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 107>**

087. react-cookies remove 사용하기

**• 학습 내용:** react-cookies의 remove 함수 사용 방법을 이해한다.

**• 힌트 내용:** remove 함수 호출 시 웹 브라우저에서 쿠키 값을 삭제할 수 있다.

remove 함수는 웹 브라우저에 남아 있는 쿠키에 key로 접근해 쿠를 삭제하는 함수다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | import React from 'react';  import CookieRemove from './R087\_cookieRemove'  function App() {  return (  <div>  <h1>Start React 200!</h1>  <CookieRemove/>  </div>  );  }  export default App; |

2: App.js 파일과 같은 경로에 위치한 R087\_cookieRemove.js 파일을 임포트해 사용할 수 있도록 한다.

src 폴더에 R087\_cookieRemove.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R087\_cookieRemove.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, { Component } from 'react';  import cookie from 'react-cookies';  class R087\_cookieRemove extends Component {  componentDidMount() {  const expires = new Date()  expires.setMinutes(expires.getMinutes() + 60)  cookie.save('userid', "react200"  , {  path: '/',  expires,  // secure: true,  // httpOnly: true  }  );  setTimeout(function() {  cookie.remove('userid', { path: '/'});  },1000);  setTimeout(function() {  alert(cookie.load('userid'))  },2000);  }  render() {  return (  <><h3>react-cookies Remove</h3></>  )  }  }  export default R087\_cookieRemove; |

16~18: line 8에서 쿠키 저장 코드가 실행되고 1초 후에 cookie.remove 함수를 사용해 루트(/) 경로에 저장된 userid라는 쿠키 값을 삭제한다.

19~21: cookie.remove 함수가 실행되고 약 1초 후 웹 브라우저에 저장된 쿠키 값을 alert으로 확인한다. 실행 결과를 보면 undefined로 쿠키가 삭제된 것을 확인할 수 있다.

|  |
| --- |
| NOTE |
| 예제에서 흐름상 cookie.save가 완료되고 cookie.remove, cookie.remove가 완료되고 cookie.load가 실행돼야 한다. 그런데 자바스크립트는 비동기적으로 동작하기 때문에 이 순서를 보장할 수 없다. 이때 setTimeout 함수를 사용하면 대략적인 완료 시간을 확보할 수 있다. |

<실행 결과>

|  |
| --- |
|  |

**<그림 108>**

088. react-router-dom BrowserRouter 사용하기

**• 학습 내용:** react-router-dom 패키지의 BrowserRouter 사용 방법을 이해한다.

**• 힌트 내용:** 라우팅 개념을 이해하고 <BrowserRouter> 태그의 역할을 확인한다.

라우팅(routing)이란, 호출되는 url에 따라 페이지(view)이동을 설정하는 것이다. react에서 view는 component를 사용한다. react에서 라우팅을 지원하는 패키지는 여러 개 있지만, 웹 개발을 위한 패키지로 적합한 react-router-dom을 설치한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install --save react-router-dom]를 입력하면 다음과 같이 npm이 react-router-dom을 설치한다.

|  |
| --- |
|  |

**<그림 109>**

react-router-dom 패키지는 BrowserRouter, Route, Link의 기능을 제공한다. Route는 호출되는 url에 따라 이동할 component를 정의한다. Link는 <a> 태그와 같이 페이지에 표시되는 링크를 클릭하면 url을 호출한다. Route와 Link를 사용하기 위해서는 <BrowserRouter> 태그로 감싸 사용해야 한다.

이후 예제에서 component들을 폴더 단위로 묶어서 관리하기 위해 파일 구조를 다음과 같이 변경한다.

|  |
| --- |
|  |

**<그림 110>**

index.js 파일을 다음과 같이 수정한다.

<소스> index.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React from 'react';  import ReactDOM from 'react-dom';  import { BrowserRouter } from 'react-router-dom';  import './index.css';  import App from './components/App';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render((  <BrowserRouter>  <App />  </BrowserRouter>  ), document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

3: react-router-dom 패키지의 BrowserRouter를 임포트해 사용할 수 있도록 한다.

9~11: App.js는 라우팅 전용 컴포넌트로 사용될 것이다. App 컴포넌트와 하위 컴포넌트에서 Route, Link를 사용할 때마다 <BrowserRouter> 태그를 감싸 사용할 수도 있다. 하지만 최상위 컴포넌트인 App 컴포넌트를 <BrowserRouter> 태그 안에 위치시키면, 하위 모든 컴포넌트에서 라우팅 기능(Route, Link)을 사용할 수 있다.

88~89번의 실행 결과는 89번에서 확인한다.

089. react-router-dom Route 사용하기

**• 학습 내용:** react-router-dom 패키지의 Route 사용 방법을 이해한다.

**• 힌트 내용:** 라우팅 개념을 이해하고 <Route> 태그의 역할을 확인한다.

Route는 서버에 호출된 url의 path에 따라 연결할 component를 정의한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  import { Route } from "react-router-dom";  import reactRouter from './R089\_reactRouter'  import reactRouter2 from './R089\_reactRouter2'  class App extends Component {  render () {  return (  <div className="App">  <Route exact path='/' component={reactRouter} />  <Route exact path='/reactRouter2' component={reactRouter2} />  </div>  );  }  }  export default App; |

3: react-router-dom을 임포트해 Route 기능을 사용할 수 있도록 한다.

10: Route> 태그에 path 속성은 호출되는 url 경로, component 속성은 연결할 컴포넌트를 할당한다. Line 3에서 임포트한 R089\_reactRouter 컴포넌트를 루트 경로(/)에 연결한다.

11: line 4에서 임포트한 R089\_reactRouter2 컴포넌트를 /reactRouter2 경로에 연결한다.

components 폴더에 R089\_reactRouter.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R089\_reactRouter.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14 | import React, { Component } from 'react';  class R089\_reactRouter extends Component {  render() {  return (  <>  <h1>path='/'</h1>  <h3>R089\_reactRouter</h3>  </>  )  }  }  export default R089\_reactRouter; |

7~8: 단순히 path별로 라우팅이 잘됐는지 확인하기 위해 path와 컴포넌트명을 화면에 출력한다. 루트 경로(/)로 호출했을 때 연결되는 R089\_reactRouter 컴포넌트다.

<실행 결과>

|  |
| --- |
|  |

**<그림 111>**

components 폴더에 R089\_reactRouter2.js 파일을 생성한 후 다음과 같이 입력한다.

<소스> R089\_reactRouter2.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14 | import React, { Component } from 'react';  class R089\_reactRouter2 extends Component {  render() {  return (  <>  <h1>path='/reactRouter2'</h1>  <h3>R089\_reactRouter</h3>  </>  )  }  }  export default R089\_reactRouter2; |

7~8: 단순히 path별로 라우팅이 잘됐는지 확인하기 위해 path와 컴포넌트명을 화면에 출력한다. /reactRouter2 경로로 호출했을 때 연결되는 R089\_reactRouter2 컴포넌트다.

<실행 결과>

|  |
| --- |
|  |

**<그림 112>**

090. react-router-dom Link 사용하기

**• 학습 내용:** react-router-dom 패키지의 Link 사용 방법을 이해한다.

**• 힌트 내용:** 라우팅 개념을 이해하고 <Link> 태그의 역할을 확인한다.

Link는 <a> 태그와 동일하게 동작한다. <Route> 태그에 정의한 path를 Link 속성에 연결해 놓으면, 링크를 클릭했을 때 라우팅된 컴포넌트로 이동한다.

components 폴더에 R089\_reactRouter.js 파일을 다음과 같이 수정한다.

<소스> R089\_reactRouter.js

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7  8  9  10  11  12  13  14  15  16 | import React, { Component } from 'react';  import { Link } from 'react-router-dom'  class R089\_reactRouter extends Component {  render() {  return (  <>  <h1>path='/'</h1>  <h3>R089\_reactRouter</h3>  <Link to={'/reactRouter2'}>reactRouter2</Link>  </>  )  }  }  export default R089\_reactRouter; |

2: react-router-dom 패키지를 임포트해 Link 기능을 사용할 수 있도록 한다.

10: <Link> 태그를 추가하고 to 속성에 연결할 path를 입력한다. 이때 path는 App.js의 <Route> 태그에서 특정 컴포넌트로 라우팅 처리가 돼 있어야 한다. 개발자 도구의 [Elements] 탭에서 <Link> 태그의 코드를 보면, 실행 결과와 같이 <a> 태그인 것을 확인할 수 있다.

<실행 결과>

|  |
| --- |
|  |

**<그림 113>**

091. header, footer 구현하기

**• 학습 내용:** header, footer를 구현하는 방법을 이해한다.

**• 힌트 내용:** header, footer는 공통 페이지이기 때문에 언제나 표시돼야 한다.

react-router-dom 패키지의 Route 기능을 사용하면, 호출되는 url에 따라 서로 다른 컴포넌트를 표시할 수 있었다. 하지만 header와 footer는 라우팅과 상관없이 항상 표시돼야 하는 영역이다.

다음과 같이 components 폴더에 Footer와 Header 폴더를 만든 후 그 안에 각각의 컴포넌트를 생성해 사용한다. src 경로에 css와 img 폴더를 생성한다. 그리고 생성한 폴더 안에 예제에서 사용할 css와 이미지 파일을 추가한다.

|  |
| --- |
|  |

**<그림 114>**

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, { Component } from 'react';  import { Route } from "react-router-dom";  import reactRouter from './R089\_reactRouter'  import reactRouter2 from './R089\_reactRouter2'  // css  import '../css/new.css';  // header  import HeaderAdmin from './Header/Header admin';  // footer  import Footer from './Footer/Footer';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={reactRouter} />  <Route exact path='/reactRouter2' component={reactRouter2} />  <Footer/>  </div>  );  }  }  export default App; |

7: 라우팅 컴포넌트인 App.js에서 사용할 css 파일을 임포트하면, 하위 모든 컴포넌트에 css 파일이 적용된다. 파일 경로 앞에 점이 한 개(.)만 붙으면, 현재 파일(App.js)이 있는 폴더 경로(components)를 의미한다. 점이 두 개(..)라면 한 단계 상위 폴더로 이동한다. 예제에서는 App.js가 있는 components보다 한 단계 상위 폴더인 src에서 css라는 폴더에 접근한다.

19: line 10에서 임포트한 header 컴포넌트를 <Route> 태그보다 위쪽에 위치시킨다.

19: line 13에서 임포트한 footer 컴포넌트를 <Route> 태그보다 아래쪽에 위치시킨다.

실행 결과를 보면, 상단에 header가 하단에 footer 영역이 고정으로 위치한다. 그리고 중간 영역에 라우팅되는 컴포넌트가 표시된다. 루트 경로(/)에서 /reactRouter2 경로로 url을 이동해도 중간에 표시되는 컴포넌트 영역만 변경된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 115>**

092. home 페이지 만들기

**• 학습 내용:** home 페이지를 구현하는 방법을 이해한다.

**• 힌트 내용:** home 페이지는 처음 웹 사이트에 접속하면 표시되는 화면이다.

home 페이지는 루트 경로(/)로 url을 호출했을 때 접속하게 되는 페이지다. home 화면을 구성하는 방법은 다양한데, 예제에서는 로그인 페이지를 표시한다.

App.js 파일을 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | import React, { Component } from 'react';  import { Route } from "react-router-dom";  // css  import '../css/new.css';  // header  import HeaderAdmin from './Header/Header admin';  // footer  import Footer from './Footer/Footer';  // login  import LoginForm from './LoginForm';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Footer/>  </div>  );  }  }  export default App; |

14: 로그인 화면을 나타내는 LoginForm 컴포넌트를 임포트한다.

21: 루트 경로(/)로 웹 사이트에 접속하면, line 14에서 임포트한 LoginForm 컴포넌트로 라우팅한다.

<실행 결과>

|  |
| --- |
|  |

**<그림 116>**

093. <react img> 태그 사용하기

**• 학습 내용:** react에서 <img> 태그를 사용하는 방법을 이해한다.

**• 힌트 내용:** react 서버 경로에 이미지를 추가하면, 화면에 표시시킬 수 있다.

html에서 <img> 태그에 src 속성 값으로 이미지 경로를 직접 할당하면, 화면에 이미지를 표시할 수 있다. react에서도 동일한 구조로 사용하지만, require 문법을 사용해 이미지 경로를 할당해야 한다.

로그인 폼 컴포넌트인 LoginForm.js 파일에서 <img> 태그 사용 방법을 확인한다.

<소스> LoginForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | import React, { Component } from 'react';  import { Link } from 'react-router-dom'  class LoginForm extends Component {  render () {  return (  <section className="main">  <div className="m\_login">  <h3><span><img src={require("../img/main/log\_img.png")} alt="" />  </span>LOGIN</h3>  <div className="log\_box">  <form>  <div className="in\_ty1">  <span><img src={require("../img/main/m\_log\_i3.png")} alt="" /></span>  <input type="text" id="email\_val" name="email" placeholder="이메일" />  </div>  <div className="in\_ty1">  <span className="ic\_2">  <img src={require("../img/main/m\_log\_i2.png")} alt="" />  </span>  <input type="password" placeholder="비밀번호" />  </div>  <ul className="af">  <li><Link to={'/register\_check'}>회원 가입</Link></li>  <li className="pwr\_b"><a href="#n">비밀번호 재설정</a></li>  </ul>  <button className="s\_bt" type="submit">로그인</button>  </form>  </div>  </div>  </section>  );  }  }  export default LoginForm; |

9, 14, 19: <img> 태그의 src 속성 값을 require 문법을 사용해 할당한다. 경로에 점(.)이 2개이기 때문에 LoginForm.js 파일이 있는 폴더의 한 단계 상위 폴더인 src 경로에서 img/main 폴더를 찾는다. main 폴더를 보면 다음과 같이 사용하려는 이미지가 존재한다.

|  |
| --- |
|  |

**<그림 117>**

<실행 결과>

|  |
| --- |
|  |

**<그림 118>**

094. lodash 디바운스 사용하기

**• 학습 내용:** lodash 패키지의 debounce 함수 사용 방법을 이해한다.

**• 힌트 내용:** 검색어 자동 완성 기능을 debounce로 구현할 수 있다.

debounce는 연속된 이벤트 호출이 일어나는 상황에 사용한다. 마지막 이벤트가 실행되고 일정 시간 동안, 추가 이벤트가 발생하지 않을 때 실행되는 함수다. debounce는 꼭 필요한 시점에만 함수를 실행해 서버 자원을 효율적으로 사용할 수 있게 해준다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install --save lodash]를 입력하면 다음과 같이 npm이 lodash를 설치한다.

|  |
| --- |
|  |

**<그림 119>**

components 폴더의 R094\_reactDebounce.js 파일을 확인한다.

<소스> R094\_reactDebounce.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  import { debounce } from "lodash";  class R094\_reactDebounce extends Component {  debounceFunc = debounce(() => {  console.log("Debounce API Call");  }, 1000);    render() {  return (  <>  <h2>검색어 입력</h2>  <input type="text" onChange={this.debounceFunc} />  </>  )  }  }  export default R094\_reactDebounce; |

2: lodash 패키지를 임포트해 debounce 함수를 사용할 수 있도록 한다.

13: <input> 태그에 텍스트를 입력할 때마다 debounceFunc 함수를 호출한다.

5~7: debounce 함수에 1초의 지연 시간을 할당한다. debounceFunc 함수는 글자가 입력될 때마다 호출되지만, debounce 함수는 마지막 호출이 끝나고 1초 후에 콘솔 로그를 출력한다.

실제로 사용자가 입력한 텍스트가 포함된 검색어를 불러와야 할 때 line 6에 검색어 데이터를 호출하는 코드가 위치해야 한다. 이때 디바운스를 사용하지 않고 react라는 글자를 입력한다면 r, re, rea, reac, react가 입력되는 시점에 모두 데이터를 호출해야 한다. 디바운스를 사용하면 react라는 글자가 모두 작성되고 지연시간 1초가 지난 후 데이터를 한 번만 호출한다.

R094\_reactDebounce 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /Debounce 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import reactDebounce from './R094\_reactDebounce';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Route exact path='/Debounce' component={reactDebounce} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 120>**

095. lodash 스로틀 사용하기

**• 학습 내용:** lodash 패키지의 throttle 함수 사용 방법을 이해한다.

**• 힌트 내용:** 검색어 자동 완성 기능을 throttle로 구현할 수 있다.

throttle도 debounce와 동일하게 시간조건을 추가해 실행 횟수를 제한한다. 차이점은 debounce가 연속된 이벤트 중 마지막 이벤트가 발생한 시점부터 특정시간이 지났을 때 동작한다면, throttle는 발생한 이벤트 수와 관계없이 지정한 시간 당위당 최대 한 번만 동작한다.

components 폴더의 R095\_reactThrottle.js 파일을 확인한다.

<소스> R095\_reactThrottle.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | import React, { Component } from 'react';  import { throttle } from "lodash";  class R095\_reactThrottle extends Component {  throttleFunc = throttle(() => {  console.log("Throttle API Call");  }, 1000);    render() {  return (  <>  <h2>검색어 입력</h2>  <input type="text" onChange={this.throttleFunc} />  </>  )  }  }  export default R095\_reactThrottle; |

2: lodash 패키지를 임포트해 throttle 함수를 사용할 수 있도록 한다.

13: <input> 태그에 텍스트를 입력할 때마다 throttleFunc 함수를 호출한다.

5~7: throttle 함수에 1초의 실행 시간 간격을 할당한다. throttleFunc 함수는 글자가 입력될 때마다 호출되지만, throttle 함수는 1초마다 0개 또는 1개의 로그를 출력한다.

실제로 사용자가 입력한 텍스트가 포함된 검색어를 불러와야 할 때 line 6에 검색어 데이터를 호출하는 코드가 위치해야 한다. 만약 react라는 검색어를 0초(r), 0.3초(e), 0.5초(a), 1.1초(c), 2초(t)에 입력했다면, throttle는 0초(r), 1초(rea), 2초(react)에 1초 단위로 데이터 호출 코드를 실행한다.

R095\_reactThrottle 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /Throttle 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import reactThrottle from './R095\_reactThrottle';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Route exact path='/Throttle' component={reactThrottle} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 121>**

(도비라)4부 활용: <저자 요청> 제목 요망(2부 도비라 참조)

**<저자 요청: 5~6줄 정도의 발문 부탁드립니다.>**

096. 서울시 유동 인구 데이터 사용하기 1 - 리스트 페이지 만들기

**• 학습 내용:** 리스트 페이지 구현 방법을 이해한다.

**• 힌트 내용:** 사용할 데이터를 호출하기 전에 하드코딩으로 리스트 레이아웃을 확인한다.

유동 인구 데이터 리스트를 표시할 컴포넌트를 추가한다. 실제 데이터를 불러와 사용하기 전에 데이터를 하드코딩으로 작성하고 미리 페이지 레이아웃을 확인한다.

Components > Floating\_population 폴더의 floatingPopulationList.js 파일을 확인한다.

<소스> floatingPopulationList.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | import React, { Component } from 'react';  class floatingPopulationList extends Component {  render () {  return (  .. 코드 생략..  <table class="table\_ty1 fp\_tlist">  <tr>  <th>Row</th>  <th>일자</th>  <th>시간</th>  <th>연령대</th>  <th>성별</th>  <th>시</th>  <th>군구</th>  <th>유동 인구 수</th>  </tr>  </table>  <table class="table\_ty2 fp\_tlist">  <tr class="hidden\_type">  <td>1</td>  <td>20191101</td>  <td>00</td>  <td>40</td>  <td>여성</td>  <td>서울</td>  <td>영등포구</td>  <td>32670</td>  </tr>  <tr class="hidden\_type">  <td>1</td>  <td>20191101</td>  <td>00</td>  <td>50</td>  <td>남성</td>  <td>서울</td>  <td>구로구</td>  <td>27888</td>  </tr>  </table>  .. 코드 생략.. |

7~18: <table> 태그에 <tr> 태그로 행을 추가하고 행 안에 <th> 태그로 table 헤더 값(컬럼명)들을 작성한다.

19~40: <table> 태그에 <tr> 태그로 행을 추가하고 행 안에 <td> 태그로 각 컬럼에 맞는 데이터를 작성한다.

floatingPopulationList 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /floatPopulationList 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import floatingPopulationList from './Floating\_population/floatingPopulationList';  class App extends Component {  render () {  return (  <div className="App">  .. 코드 생략..  <Route path='/floatPopulationList' component={floatingPopulationList} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 122>**

097. 서울시 유동 인구 데이터 사용하기 2 - Open Api Key 발급받기

**• 학습 내용:** Open Api Key를 발급받는 방법을 이해한다.

**• 힌트 내용:** 외부 Api로 데이터 호출을 위해 Open Api Key를 발급받는다.

Sk텔레콤 빅 데이터 허브([https://www.bigdatahub.co.kr](https://www.bigdatahub.co.kr/))에서 제공하는 데이터를 사용하기 위해 먼저 회원 가입을 하고 [MyPage] – [기본 정보 관리] – [API Access Key 관리]에서 다음과 같이 API ACCESS KEY를 확인한다.

|  |
| --- |
|  |

**<그림 123>**

예제에서 사용할 데이터는 [서울시 유동 인구 데이터 – 19년 11월]로 [www.bigdatahub.co.kr/product/view.do?pid=1002277](http://www.bigdatahub.co.kr/product/view.do?pid=1002277)로 접속해 샘플 데이터와 API 정보를 확인할 수 있다. 다음 화면 보이는 요청 주소(<https://www.bigdatahub.co.kr/product/view.do?pid=1002277>)를 크롬 웹 브라우저에서 그대로 호출하면 클라이언트 인증 에러 메시지가 확인된다.

|  |
| --- |
| **<그림 124>** |

URL에 API ACCESS KEY를 추가해야 정상적으로 데이터를 받을 수 있다. 다음 URL처럼 TDCAccessKey 파라미터에 키 값을 할당해 API를 호출해야 한다.

https://api.bigdatahub.co.kr/v1/datahub/datasets/search.json?pid=1002277&TDCAccessKey={API ACCESS KEY키}

추가로 [API 연동 규석서 다운로드]를 받아보면 구체적인 API 사용 방법을 확인할 수 있다.

098. 서울시 유동 인구 데이터 사용하기 3 - JSON Viewer로 데이터 확인하기

**• 학습 내용:** 크롬 확장 프로그램 JSON Viewer로 데이터를 확인한다.

**• 힌트 내용:** react 서버에서 Api를 호출하기 전에 웹 브라우저에서 데이터 형태를 확인한다

예제에서 사용할 Api는 get 방식으로 호출할 수 있고 json 형태로 데이터가 반환할 수도 있다. 크롬 웹 브라우저에서 가독성 있게 json 데이터를 확인하기 위해 JSON Viewer라는 확장 프로그램을 설치한다.

|  |
| --- |
| **<그림 125>** |

JSON View 설치 후 크롬 웹 브라우저에서 URL을 호출해보면, 다음과 같이 정렬된 json 데이터를 확인할 수 있다.

|  |
| --- |
| **<그림 126>** |

099. 서울시 유동 인구 데이터 사용하기 4 - axios로 Api 호출하기

**• 학습 내용:** axios로 외부 Api를 호출하는 방법을 확인한다.

**• 힌트 내용:** axios로 호출한 유동 인구 데이터를 리스트 화면에 표시한다.

axios 함수를 사용해 get 방식으로 유동 인구 Api를 호출하고 데이터를 반복문을 사용해 행 단위의 리스트로 생성한다.

[Components] – [Floating\_population] 폴더의 floatingPopulationList.js 파일을 확인한다.

<소스> floatingPopulationList.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | import React, { Component } from 'react';  import axios from "axios";  class floatingPopulationList extends Component {  constructor(props) {  super(props);  this.state = {  responseFPList: '',  append\_FPList: '',  }  }  componentDidMount() {  this.callFloatPopulListApi()  }  callFloatPopulListApi = async () => {  axios.get('https://api.bigdatahub.co.kr/v1/datahub/datasets/search.json?'+  'pid=1002277&TDCAccessKey=키값&$count=30', {  })  .then( response => {  try {  this.setState({ responseFPList: response });  this.setState({ append\_FPList: this.FloatPopulListAppend() });  } catch (error) {  alert(error)  }  })  .catch( error => {alert(error);return false;} );  }  FloatPopulListAppend = () => {  let result = []  var FPList = this.state.responseFPList.data  var jsonString = JSON.stringify(FPList)  jsonString = jsonString.replace(/\(1시간단위\)/g, '')  jsonString = jsonString.replace(/\(10세단위\)/g, '')  var json = JSON.parse(jsonString)  for(let i=0; i<json.entry.length; i++){  var data = json.entry[i]  var idx = i+1  result.push(  <tr class="hidden\_type">  <td>{idx}</td>  <td>{data.일자}</td>  <td>{data.시간}</td>  <td>{data.연령대}</td>  <td>{data.성별}</td>  <td>{data.시}</td>  <td>{data.군구}</td>  <td>{data.유동 인구 수}</td>  </tr>  )  }  return result  }  render () {  return (  .. 코드 생략..  <table class="table\_ty2 fp\_tlist">  {this.state.append\_FPList}  </table>  .. 코드 생략.. |

2: axios를 임포트해 사용할 수 있도록 한다.

9~10: constructor 함수가 가장 먼저 실행된다. 호출된 response 값을 저장할 responseFPList 변수와 리스트 html 코드를 저장할 append\_FPList 변수를 state로 선언하고 공백을 할당한다.

64: constructor 함수 다음에 render 함수가 실행된다. render 함수가 최초 실행될 때는 append\_FPList가 공백으로 빈 테이블 상태다.

14~16: render 함수 다음으로 componentDidMount 함수가 실행된다. callFloatPopulListApi를 호출한다.

19~20: callFloatPopulListApi 함수가 실행되면, axios로 유동 인구 url을 호출한다. 파라미터로 $count=30를 추가하면 30개의 데이터를 호출할 수 있다.

24: axios 호출로 반환된 response 데이터를 state 변수 responseFPList에 할당한다.

25: FloatPopulListAppend 함수 호출한다.

36~39: FloatPopulListAppend 함수가 호출되면, responseFPList 변수에 할당했던 json 형태의 데이터를 불러온다. json 데이터를 string으로 변환해 불필요한 텍스트를 공백으로 replace한다.

41~56: json 데이터에서 실제 사용할 데이터가 포함된 entry object에 접근한다. json 데이터를 순서대로 가져온 후 <tr>, <td> 태그로 감싸 테이블의 한 행을 만들어준다. 행을 하나 생성할 때마다 line 34에서 선언한 배열 result에 push 문법으로 삽입한다. 반목문으로 모든 행을 result 배열에 넣고 반환한다.

25: line 57에서 return된 행들(<tr> 태그 묶음)을 append\_FPList에 할당한다. 이때 setState 함수가 실행되면서 render 함수가 다시 한번 호출된다.

64: render 함수가 다시 실행될 시점에 append\_FPList에 할당된 html 코드(<tr> 태그 묶음)이 화면에 표시된다.

<실행 결과>

|  |
| --- |
|  |

**<그림 127>**

100. recharts로 LineChart 구현하기 1

**• 학습 내용:** recharts 패키지의 LineChart 구현 방법을 이해한다.

**• 힌트 내용:** recharts 패키지의 태그에 형식에 맞는 데이터를 넣으면, 차트를 시각화할 수 있다.

recharts 패키지는 react에서 간편하게 다양한 형태의 chart를 구현할 수 있도록 api를 제공한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install --save recharts]를 입력하면 다음과 같이 npm이 recharts를 설치한다.

|  |
| --- |
|  |

**<그림 128>**

Floating\_population 폴더의 rechartsSimpleLineChart.js 파일을 확인한다.

<소스> rechartsSimpleLineChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import React, { PureComponent } from 'react';  import {  LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend,  } from 'recharts';  const data = [  { 군구: '광진구', 유동 인구 수: 32760, 비유동 인구 수: 34000 },  { 군구: '동대문구', 유동 인구 수: 30480, 비유동 인구 수: 56000 },  { 군구: '마포구', 유동 인구 수: 27250, 비유동 인구 수: 23000 },  { 군구: '구로구', 유동 인구 수: 49870, 비유동 인구 수: 67000 },  { 군구: '강남구', 유동 인구 수: 51420, 비유동 인구 수: 55000 },  ];  export default class rechartsSimpleLineChart extends PureComponent {  static jsfiddleUrl = 'https://jsfiddle.net/alidingling/xqjtetw0/';  render() {  return (  <LineChart  width={1000}  height={300}  data={data}  margin={{ top: 5, right: 30, left: 20, bottom: 5 }}  >  <CartesianGrid strokeDasharray="3 3" />  <XAxis dataKey="군구" /><YAxis />  <Tooltip />  <Legend />  <Line type="monotone" dataKey="유동 인구 수" stroke="#8884d8" activeDot={{ r: 8 }} />  <Line type="monotone" dataKey="비유동 인구 수" stroke="#82ca9d" />  </LineChart>  );  }  } |

2~4: recharts 패키지를 임포트해 <LineChart>, <Line>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip>, <Legend> 태그를 사용할 수 있도록 한다.

6~12: 군 구별, 유동 인구 수와 비유동 인구 수를 비교하기 위해 json 형태의 데이터를 세팅한다.

19~24: <LineChart> 태그에 화면에 표시할 차트 영역의 가로 길이(width), 세로 길이(height), 데이터(data), margin 값을 할당한다.

25: 차트 내부에 표시되는 격자선 간격을 조정할 수 있다.

26: data 변수에 할당한 데이터 중 X축에 사용할 데이터의 key 값을 지정한다.

27: <Tooltip> 태그는 마우스가 차트로 이동했을 때 이동한 좌표의 데이터를 화면에 나타낸다.

28: <Legend> 태그는 차트 하단 범례를 영역에 표시한다.

29~30: Y축에 표현될 데이터 key 값과 라인색을 지정한다. activeDot의 r 값은 마우스 커서가 차트로 이동했을 때 나타나는 색이 채워지는 동그라미의 크기다.

rechartsSimpleLineChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /rechartsSimpleLineChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import rechartsSimpleLineChart from './Floating\_population/rechartsSimpleLineChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/rechartsSimpleLineChart' component={rechartsSimpleLineChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 129>**

101. recharts로 LineChart 구현하기 2(서울시 유동 인구 데이터)

**• 학습 내용:** 유동 인구 api 호출 데이터로 LineChart를 구현한다.

**• 힌트 내용:** api 호출 데이터를 LineChart에서 사용할 수 있도록 할당한다.

외부 api로 호출해서 가져온 데이터로 LineChart를 구현할 수 있다. 사용 가이드에 맞게 json 데이터를 할당하고 X, Y축 데이터를 지정한다.

Floating\_population 폴더의 floatingPopulationListChart.js 파일을 확인한다.

<소스> floatingPopulationListChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | .. 코드 생략..  class floatingPopulationListChart extends Component {  constructor(props) {  super(props);  this.state = {  responseFPList: '',  append\_FPList: '',  }  }  componentDidMount = async () => {  axios.get('https://api.bigdatahub.co.kr/v1/datahub/datasets/search.json?pid=1002277&'+  'TDCAccessKey=키값&$count=5', {  })  .then( response => {  try {  this.setState({ responseFPList: response });  this.setState({ append\_FPList: this.state.responseFPList.data.entry });  } catch (error) {  alert(error)  }  })  .catch( error => {alert(error);return false;} );  }  render () {  return (  <LineChart  .. 코드 생략..  data={this.state.append\_FPList}  .. 코드 생략..  <XAxis dataKey="군구" />  .. 코드 생략..  <Line type="monotone" dataKey="유동 인구 수" stroke="#8884d8" />  </LineChart>  .. 코드 생략.. |

2~4: recharts 패키지를 임포트해 <LineChart>, <Line>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip>, <Legend> 태그를 사용할 수 있도록 한다.

12~13: 유동 인구 api를 호출한다. 간단한 구조의 차트 구현을 위해 count 파라미터 값을 5로 할당해 5개의 데이터만 반환되도록 한다.

17: api 호출 결과 반환된 response 데이터를 state 변수 responseFPList에 할당한다.

18: LineChart에서 사용하는 데이터 형식에 맞게 json 데이터의 entry 변수까지 접근해 state 변수 append\_FPList에 할당한다.

30: line 18에서 데이터를 할당한 append\_FPList 변수를 LineChart 차트에서 사용할 수 있도록 한다.

32: X축 데이터의 key 값을 지정한다.

34: Y축 데이터의 key 값을 지정한다.

floatingPopulationListChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /floatingPopulationListChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import floatingPopulationListChart from './Floating\_population/floatingPopulationListChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/floatingPopulationListChart' component={floatingPopulationListChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 130>**

102. recharts로 AreaChart 구현하기(서울시 유동 인구 데이터)

**• 학습 내용:** 유동 인구 api 호출 데이터로 AreaChart를 구현한다.

**• 힌트 내용:** api 호출 데이터를 AreaChart에서 사용할 수 있도록 할당한다.

외부 api로 호출해서 가져온 데이터로 AreaChart를 구현할 수 있다. 사용 가이드에 맞게 json 데이터를 할당하고 X, Y축 데이터를 지정한다.

Floating\_population 폴더의 floatingPopulationAreaChart.js 파일을 확인한다.

<소스> floatingPopulationAreaChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | import React, { Component } from 'react';  import { AreaChart, Area, XAxis, YAxis, CartesianGrid, Tooltip } from 'recharts';  import axios from "axios";  class floatingPopulationAreaChart extends Component {  .. 코드 생략..  render () {  return (  <AreaChart  width={1000}  height={300}  data={this.state.append\_FPList}  margin={{  top: 5, right: 50, left: 20, bottom: 5,  }}  >  <CartesianGrid strokeDasharray="3 3" />  <XAxis dataKey="군구" />  <YAxis />  <Tooltip />  <Area type="monotone" dataKey="유동 인구 수" stroke="#8884d8" fill="#8884d8" />  </AreaChart>  .. 코드 생략.. |

2: recharts 패키지를 임포트해 <AreaChart>, <Area>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip> 태그를 사용할 수 있도록 한다.

6: 유동 인구 api를 호출해 state 변수 append\_FPList에 할당하는 코드는 LineChart와 동일하므로 생략한다.

9: <AreaChart> 태그를 사용하면, LineChart에서 line 하단 부분 색이 채워진 차트가 표시된다.

21: <Area> 태그의 dataKey 속성 값으로 X 좌표의 키 값을 할당한다.

floatingPopulationAreaChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /floatingPopulationAreaChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import floatingPopulationAreaChart from './Floating\_population/floatingPopulationAreaChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/floatingPopulationAreaChart' component={floatingPopulationAreaChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 131>**

103. recharts로 BarChart 구현하기(서울시 유동 인구 데이터)

**• 학습 내용:** 유동 인구 api 호출 데이터로 BarChart를 구현한다.

**• 힌트 내용:** api 호출 데이터를 BarChart에서 사용할 수 있도록 할당한다.

외부 api로 호출해서 가져온 데이터로 BarChart를 구현할 수 있다. 사용 가이드에 맞게 json 데이터를 할당하고 X, Y축 데이터를 지정한다.

Floating\_population 폴더의 floatingPopulationBarChart.js 파일을 확인한다.

<소스> floatingPopulationBarChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | import React, { Component } from 'react';  import { BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';  import axios from "axios";  class floatingPopulationBarChart extends Component {  .. 코드 생략..  render () {  return (  <BarChart  width={1000}  height={300}  data={this.state.append\_FPList}  margin={{  top: 5, right: 50, left: 20, bottom: 5,  }}  >  <CartesianGrid strokeDasharray="3 3" />  <XAxis dataKey="군구" />  <YAxis />  <Tooltip />  <Legend />  <Bar dataKey="유동 인구 수" fill="#82ca9d" />  </BarChart>  .. 코드 생략.. |

2: recharts 패키지를 임포트해 <BarChart>, <Bar>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip>, <Legend> 태그를 사용할 수 있도록 한다.

6: 유동 인구 api를 호출해 state 변수 append\_FPList에 할당하는 코드는 LineChart와 동일하므로 생략한다.

9: <BarChart> 태그를 사용하면 색이 채워진 막대그래프 차트가 표시된다.

22: <Bar> 태그의 dataKey 속성 값으로 X 좌표의 키 값을 할당한다. fill 속성에 막대그래프 색상 코드를 지정할 수 있다.

floatingPopulationBarChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 /floatingPopulationBarChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import floatingPopulationBarChart from './Floating\_population/floatingPopulationBarChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/floatingPopulationBarChart' component={floatingPopulationBarChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 132>**

104. recharts로 ComposedChart 구현하기(서울시 유동 인구 데이터)

**• 학습 내용:** 유동 인구 api 호출 데이터로 ComposedChart를 구현한다.

**• 힌트 내용:** api 호출 데이터를 ComposedChart에서 사용할 수 있도록 할당한다.

외부 api로 호출해서 가져온 데이터로 ComposedChart를 구현할 수 있다. ComposedChart는 여러 유형의 차트를 함께 표시한다.

Floating\_population 폴더의 floatingPopulationComposedChart.js 파일을 확인한다.

<소스> floatingPopulationComposedChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import React, { Component } from 'react';  import { ComposedChart, Line, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';  import axios from "axios";  class floatingPopulationComposedChart extends Component {  .. 코드 생략..  render () {  return (  <ComposedChart  width={1000}  height={300}  data={this.state.append\_FPList}  margin={{  top: 5, right: 50, left: 20, bottom: 5,  }}  >  <CartesianGrid stroke="#003458" />  <XAxis dataKey="군구" />  <YAxis />  <Tooltip />  <Legend />  <Bar dataKey="유동 인구 수" fill="#82ca9d" />  <Line type="monotone" dataKey="유동 인구 수" stroke="#ff7300" />  </ComposedChart>  .. 코드 생략.. |

2: recharts 패키지를 임포트해 <BarChart>, <Bar>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip>, <Legend> 태그를 사용할 수 있도록 한다.

6: 유동 인구 api를 호출해 state 변수 append\_FPList에 할당하는 코드는 LineChart와 동일하므로 생략한다.

9: <ComposedChart> 태그를 사용하면 여러 개의 차트를 동시에 표시할 수 있다.

17: <CartesianGrid> 태그의 stroke 속성에 색상 코드를 할당하면, 격자선의 색을 지정할 수 있다.

22~23: <Line> 태그와 <Bar> 태그를 <ComposedChart> 태그 안에 위치시켜 LineChart와 BarChart를 함께 표시한다.

floatingPopulationComposedChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 / floatingPopulationComposedChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13 | .. 코드 생략..  import floatingPopulationComposedChart  from './Floating\_population/floatingPopulationComposedChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/floatingPopulationComposedChart'  component={floatingPopulationComposedChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 133>**

105. recharts로 ScatterChart 구현하기(서울시 유동 인구 데이터)

**• 학습 내용:** 유동 인구 api 호출 데이터로 ScatterChart를 구현한다.

**• 힌트 내용:** api 호출 데이터를 ScatterChart에서 사용할 수 있도록 할당한다.

외부 api로 호출해서 가져온 데이터로 ScatterChart를 구현할 수 있다. ScatterChart는 X 좌표와 Y좌표 데이터를 각각 할당해 두 좌표가 만나는 지점에 점을 표시할 수 있다.

Floating\_population 폴더의 floatingPopulationScatterChart.js 파일을 확인한다.

<소스> floatingPopulationScatterChart.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | import React, { Component } from 'react';  import { ScatterChart, Scatter, XAxis, YAxis, CartesianGrid, Tooltipip } from 'recharts';  import axios from "axios";  class floatingPopulationScatterChart extends Component {  .. 코드 생략..  render () {  return (  <ScatterChart  width={1000}  height={300}  margin={{  top: 5, right: 50, left: 20, bottom: 5,  }}  >  <CartesianGrid />  <XAxis type="number" dataKey="연령대(10세단위)" name="연령대" unit="세" />  <YAxis type="number" dataKey="유동 인구 수" name="유동 인구 수" unit="명" />  <Tooltip cursor={{ strokeDasharray: '3 3' }} />  <Scatter data={this.state.append\_FPList} fill="#003458" />  </ScatterChart>  .. 코드 생략.. |

2: recharts 패키지를 임포트해 <BarChart>, <Bar>, <XAxis>, <YAxis>, <CartesianGrid>, <Tooltip>, <Legend> 태그를 사용할 수 있도록 한다.

6: 유동 인구 api를 호출해 state 변수 append\_FPList에 할당하는 코드는 LineChart와 동일하므로 생략한다.

9: <ScatterChart> 태그를 사용하면 분산형 차트를 표시할 수 있다.

17~18: XAxis는 X축에 대한 정보를 YAxis에는 Y축에 대한 정보를 할당한다. dataKey에는 사용할 데이터의 key 값을 지정한다. name과 unit은 각각 마우스를 차트의 점 위에 올려 놓았을 때 표시되는 데이터명과 단위다.

20: Scatter에 직접 데이터를 할당한다. fill 속성에 색상 코드를 할당하면, 차트에 표시되는 점 색을 변경할 수 있다.

floatingPopulationScatterChart 컴포넌트를 확인하기 위해 App.js에서 라우팅돼 있는 / floatingPopulationScatterChart 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import floatingPopulationScatterChart from './Floating\_population/floatingPopulationScatterChart';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/floatingPopulationScatterChart' component={floatingPopulationScatterChart} />  .. 코드 생략.. |

<실행 결과>

|  |
| --- |
|  |

**<그림 134>**

(도비라)5부 실무: <저자 요청> 제목 요망(2부 도비라 참조)

**<저자 요청: 5~6줄 정도의 발문 부탁드립니다.>**

106. node.js express 프레임워크 설치

**• 학습 내용:** back-end 서버를 구현하기 위해 node.js의 express 프레임워크를 설치한다.

**• 힌트 내용:** express를 사용하면 간편하게 node 서버를 구현할 수 있다.

react에서 create-react-app을 설치해 front-end 서버를 간편하게 구동했던 것처럼 node에서는 express를 사용하면 back-end 서버 구현을 쉽게 할 수 있다.

1. react 서버 디렉터리(C:\react200\client)의 상위 디렉터리(C:\react200)에 express를 설치한다. cmd을 열어 [npm i -g express-generator] 명령어로 express를 생성하기 위한 generator를 설치한다. generator 설치 후 [express nodefolder] 명령어를 실행하면 nodefolder라는 express 프로젝트가 생성된다.

|  |
| --- |
|  |

**<그림 135>**

2. nodefolder 폴더를 보면 다음과 같이 파일 구조가 생성된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 136>**

3. nodefolder 폴더 안에 있는 폴더와 파일들을 C:\react200 경로로 옮기고 nodefolder는 삭제한다.

|  |
| --- |
|  |

**<그림 137>**

107. 원고 누락(저자 확인 요망)

**107. node 서버 구동**

**• 학습 내용 :** node express서버를 구동하는 방법을 이해한다.

**• 힌트 내용 :** node서버를 구동하는 데 필요한 패키지들을 설치 후, 서버를 시작한다.

express프로젝트를 생성하면, 기본적인 웹 프레임워크 디렉터리 구조가 생성된다. react프로젝트 구조와 마찬가지로 node경로(C:\react200)에는 package.json파일이 존재하고, 서버 구동에 필요한 패키지들이 작성되어 있다. 작성되어 있는 패키지들을 설치하면, 서버 구동이 가능하다.

1. node경로(C:\react200)에서 [npm install]명령어를 실행해, package.json에 작성된 패키지들을 설치한다. 설치가 완료되면 node경로(C:\react200)에, 설치된 패키지들이 추가된 node\_modules폴더가 생성된다.

|  |
| --- |
|  |

2. node경로(C:\react200)에서 [npm start]명령어를 실행하면, 서버를 구동할 수 있다.

|  |
| --- |
|  |

3. 브라우저에서 [localhost:3000]을 호출하면, 다음과 같이 node express서버가 구동된 것을 확인할 수 있다. express서버의 기본 포트는 3000번이다.

|  |
| --- |
|  |

108. node 서버 api 호출하기

**• 학습 내용:** node 서버 api를 호출하는 방법을 이해한다.

**• 힌트 내용:** express 프레임워크에서 기본으로 제공하는 라우팅 기능을 사용한다.

node 경로(C:\react200)의 app.js 파일은 react에서 App.js 파일과 동일한 라우팅 기능을 담당한다. express 프레임워크에서 기본으로 라우터 구조가 잡혀 있기 때문에 간편하게 api를 구현할 수 있다.

1. node.js는 back-end뿐 아니라 front-end도 구현할 수 있는 언어다. 하지만 front-end 언어로는 node보다 react가 강점이 많이 때문에 node 서버는 back-end api 서버로만 사용할 것이다. node 경로(C:\react200)에서 불필요한 front-end 관련 소스 public, views 폴더를 삭제한다.

|  |
| --- |
|  |

**<그림 141>**

2. node 경로(C:\react200)에서 app.js 파일명을 react의 App.js와 구분하기 위해 server.js로 수정한다. server.js에서 불필요한 코드를 삭제하고 다음과 같이 수정한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | var express = require('express');  var indexRouter = require('./routes/index');  var usersRouter = require('./routes/users');  var app = express();  app.use('/', indexRouter);  app.use('/users', usersRouter);  module.exports = app; |

3~4: react의 App.js에서 컴포넌트 파일을 임포트했던 것처럼 require 문법으로 라우터 파일 경로를 변수에 저장한다.

8~9: 호출하는 서버 경로에 따라 line 3~4에서 지정한 파일로 라우팅한다.

|  |
| --- |
| NOTE |
| 라우팅(routing)이란, 네트워크 주소에 따라 목적지 경로를 체계적으로 결정하는 경로 선택 과정이다. |

3. routes 폴더의 users.js 파일을 보면 /users 경로의 라우터가 구현돼 있다.

<소스> users.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | var express = require('express');  var router = express.Router();  /\* GET users listing. \*/  router.get('/', function(req, res, next) {  res.send('respond with a resource');  });  module.exports = router; |

5~6: http get 방식으로 request 호출을 받으면, res.send 함수가 텍스트 데이터를 response로 전송한다. Line 5에서 req는 request, res는 response의 약자다.

4. bin 폴더의 www 파일을 열어 다음과 같이 수정한다.

<소스> www

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10 | .. 코드 생략..  var app = require('../server');  var debug = require('debug')('nodefolder:server');  var http = require('http');  .. 코드 생략..  var port = normalizePort(process.env.PORT || '5000');  app.set('port', port);  .. 코드 생략.. |

2: app.js 파일명을 server.js로 바꿨기 때문에 server.js를 require하도록 수정한다.

8: react 서버와 node 서버를 동시에 구동하기 위해서 node 서버의 포트를 5000번으로 지정한다.

웹 브라우저에서 node 서버 주소(localhost:5000)에 users 경로를 호출하면, user.js의 line 6에서 response로 반환한 텍스트가 화면에 표시된다. 이때 <http://localhost:5000/users>가 get 방식으로 호출할 수 있는 api 주소다.

|  |
| --- |
|  |

**<그림 142>**

109. react 서버와 node 서버를 프록시로 연결하기

**• 학습 내용:** 프록시의 기능과 사용 방법을 이해한다.

**• 힌트 내용:** react 서버를 프록시 서버로 사용한다.

프록시란, 클라이언트가 다른 서버에 간접적으로 접속할 수 있도록 중계해주는 프로그램이다. 예를 들면, 웹 브라우저(클라이언트)에 react 서버(localhost:3000)을 띄우고 react 페이지에서 node api를 호출해 데이터를 표시할 수 있다. 이때 프록시를 설정하면 node 서버 url을 localhost:5000/users가 아닌 /users로 간략하게 사용할 수 있다. 프록시를 보통 보안이나 캐시 목적으로 사용하는데, 예제에서는 node 서버 호출 url을 간략히 사용하기 위한 용도로 사용한다.

client 폴더의 package.json 하단에 proxy 설정 코드를 추가한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | .. 코드 생략..  "development": [  "last 1 chrome version",  "last 1 firefox version",  "last 1 safari version"  ]  },  "proxy": "http://127.0.0.1:5000"  } |

7~8: react 서버의 package.json 파일에서 node 서버 주소(127.0.0.1:5000)를 프록시로 중계한다.

components 폴더의 R109\_reactProxy.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  class R109\_reactProxy extends Component {  componentDidMount = async () => {  const response = await fetch('/users');  const body = await response.text();  console.log("body : "+body)  }    render() {  return (  <><h1>Proxy Call Node Api</h1></>  )  }  }  export default R109\_reactProxy; |

5: 프록시 설정을 해줬기 때문에 node 서버 api 주소를 localhost:5000/users가 아닌 /users로 사용할 수 있다. api를 호출한 후 반환된 값을 response 변수에 할당한다.

6~7: response 변수의 값을 확인하기 위해 .text 함수로 문자열로 변환해 로그를 출력한다.

다음과 같이 node 경로(C:\react200)와 react 경로(C:\react200\client)에서 [npm start] 명령어로 각각 node 서버와 react 서버를 구동한다.

|  |
| --- |
| **<그림 143>**    **<그림 144>** |

R109\_reactProxy 컴포넌트의 결과를 확인하기 위해 App.js에서 라우팅돼 있는 /reactProxy 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import reactProxy from './R109\_reactProxy';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route exact path='/reactProxy' component={reactProxy} />  .. 코드 생략.. |

|  |
| --- |
|  |

**<그림 145>**

110. node 서버 Api를 get으로 json 데이터 호출하기

**• 학습 내용:** node 서버의 json 데이터를 get 방식으로 호출하는 방법을 이해한다.

**• 힌트 내용:** json 데이터의 key 값으로 접근해 value 값을 가져온다.

react에서 json 데이터를 받아 사용하기 위해서는 node 서버에서 json 형태로 데이터를 response에 담아 보내줘야 한다.

node 경로(C:\react200) routes 폴더의 users.js 파일을 확인한다.

<소스> users.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | var express = require('express');  var router = express.Router();  /\* GET users listing. \*/  router.get('/', function(req, res, next) {  res.send({'message':'node get success'});  });  module.exports = router; |

5~7: node 서버(localhost:5000)에서 users 경로를 get 방식으로 호출하면, key가 message이고 value가 node get success인 json 데이터를 response로 전송한다.

components 폴더의 R110\_ApiGetJson.js 파일을 확인한다.

<소스> R110\_ApiGetJson.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React, { Component } from 'react';  class R110\_ApiGetJson extends Component {  componentDidMount = async () => {  const response = await fetch('/users');  const body = await response.json();  console.log("body.message : "+body.message)  }    render() {  return (  <><h1>Call Node Api Get</h1></>  )  }  }  export default R110\_ApiGetJson; |

6: .text가 아닌 .json 함수로 json 형태의 데이터를 body 변수에 할당한다.

7: json 데이터의 key 값인 message로 접근해 value 값 node get success를 로그로 출력한다.

R110\_ApiGetJson 컴포넌트의 결과를 확인하기 위해 App.js에서 라우팅돼 있는 /ApiGetJson 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import ApiGetJson from './R110\_ApiGetJson';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route exact path='/ApiGetJson' component={ApiGetJson} />  .. 코드 생략.. |

|  |
| --- |
|  |

**<그림 146>**

111. node 서버 Api를 post로 json 데이터 호출하기

**• 학습 내용:** node 서버의 json 데이터를 post 방식으로 호출하는 방법을 이해한다.

**• 힌트 내용:** json 데이터의 key 값으로 접근해 value 값을 가져온다.

react에서 post 호출로 json 데이터를 받아오기 위해서는 node 서버에서 post 호출을 받는 라우터 코드를 추가해야 한다.

node 경로(C:\react200) routes 폴더의 users.js 파일을 확인한다.

<소스> users.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | var express = require('express');  var router = express.Router();  /\* Post users listing. \*/  router.post('/', function(req, res, next) {  res.send({'message':'node post success'});  });  module.exports = router; |

5~7: node 서버(localhost:5000)에서 users 경로를 post 방식으로 호출하면, key가 message이고 value가 node post success인 json 데이터를 response로 전송한다. get 방식과 동일한 구조로 라우터를 사용한다.

components 폴더의 R111\_ApiPostJson.js 파일을 확인한다.

<소스> R111\_ApiGetJson.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import React, { Component } from 'react';  const axios = require('axios');  class R111\_ApiPostJson extends Component {  componentDidMount(){  axios.post('/users', {  })  .then( response => {  console.log("response.data.message : "+response.data.message)  })  }    render() {  return (  <><h1>Call Node Api Post</h1></>  )  }  }  export default R111\_ApiPostJson; |

6: .axios를 사용해 post 방식으로 node api(/users)를 호출한다.

9: json 데이터의 key 값인 message로 접근해 value 값 node post success를 로그로 출력한다.

R111\_ApiPostJson 컴포넌트의 결과를 확인하기 위해 App.js에서 라우팅돼 있는 /ApiPostJson 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import ApiPostJson from './R111\_ApiPostJson';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route exact path='/ApiPostJson' component={ApiPostJson} />  .. 코드 생략.. |

|  |
| --- |
|  |

**<그림 147>**

112. concurrently로 react, node 서버 한 번에 구동하기

**• 학습 내용:** concurrently로 두 가지 명령어를 동시에 실행할 수 있다.

**• 힌트 내용:** concurrently로 하나의 명령어로 react, node 서버를 구동할 수 있다.

concurrently는 여러 개의 명령어를 동시에 실행할 수 있게 도와줘는 패키지다.

cmd 창을 열어 node 경로(C:\react200) 로 이동한다. [npm install --save concurrently]를 입력하면 다음과 같이 npm이 concurrently를 설치한다.

|  |
| --- |
|  |

**<그림 148>**

추가로 nodemon이라는 패키지를 설치한다. nodemon은 node 폴더의 소스가 수정될 때 자동으로 node 서버를 재시작시켜준다. [npm install -g nodemon]를 입력해 nodemon을 설치한다.

|  |
| --- |
|  |

**<그림 149>**

node 경로(C:\react200)의 package.json 파일을 확인한다.

<소스> package.json

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7 | .. 코드 생략..  "scripts": {  "client": "cd client && yarn start",  "server": "nodemon server.js",  "dev": "concurrently --kill-others-on-fail \"yarn server\" \"yarn client\""  },  .. 코드 생략.. |

5: yarn dev 명령어를 실행하면, concurrently가 yarn server와 yarn client 명령어를 동시에 실행한다. --kill-others-on-fail은 두 명령어 중 하나라도 에러가 나면, 모두 종료하는 옵션이다.

3: yarn client 명령어는 react 경로로 폴더 경로로 이동해(cd client), react 서버를 실행한다(yarn start).

4: yarn server 명령어는 node 경로에서 server.js를 nodemon으로 실행해 node 서버를 구동한다. express 기본 설정에서는 포트 정보가 있는 www 파일을 실행했지만, 수정 후 www 파일 대신 server.js를 참조하기 때문에 server.js 소스에 포트 정보를 작성해야 한다.

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12 | var express = require('express');  var indexRouter = require('./routes/index');  var usersRouter = require('./routes/users');  var app = express();  app.use('/', indexRouter);  app.use('/users', usersRouter);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

11~12: node 서버 포트를 5000으로 설정하는 코드를 추가한다.

이제 cmd 하나로 react와 node 서버를 동시에 구동할 수 있다. node 경로(C:\react200)에서 yarn dev 명령어를 실행하면 react와 node 서버가 구동된다. /ApiPostJson 경로를 호출해 react 페이지에서 node api를 정상적으로 호출한 것을 확인한다.

|  |
| --- |
|  |

**<그림 150>**

113. MYSQL 서버 구축하기 1 – AWS 가입하기

**• 학습 내용:** 웹 서버(EC2)와 DB 서버(RDS)를 사용하기 위해 AWS 계정을 생성한다.

**• 힌트 내용:** AWS 서버를 사용하면 외부에서 접속 가능한 웹 사이트를 만들 수 있다.

예제에서 필요한 서버는 react와 node 서버가 구동될 EC2라는 ubuntu 서버와 DB(mysql) 서버로 사용될 RDS 서버다. AWS에서는 지정된 한도 내에서 AWS 서비스를 무료로 사용할 수 있도록 프리티어라는 기능을 제공한다. 계정당 EC2 서버와 RDS 서버를 각각 1개씩, 특정 사양으로 1년간 무료로 사용할 수 있다.

1. AWS 웹 사이트(<https://aws.amazon.com/ko/>)에 접속해 AWS 계정 생성 페이지에서 다음과 같이 이메일 주소와 암호 등을 입력하고 [동의하고 계정 만들기] 버튼을 클릭한다.

|  |
| --- |
|  |

**<그림 151>**

2. 다음 페이지에서 계정 유형을 [개인]으로 선택한다. 나머지 정보를 입력한 후 [ 계정을 만들고 계속 진행] 버튼을 클릭한다.

|  |
| --- |
|  |

**<그림 152>**

3. 마지막 단계에서 결제 정보를 입력한다. [검증 및 추가] 버튼을 클릭했을 때 유효한 카드 정보를 입력했다면 1달러가 결제되지만 3~4일 안에 승인 취소된다. 프리티어를 사용한 후 1년이 만료되기 전에 생성한 모든 인스턴스를 삭제하면 과금이 될 일은 없다. 과금이 불안하다면 등록한 카드를 분실 신고하는 등의 방법으로 결제되지 않게 할 수 있다.

|  |
| --- |
|  |

**<그림 153>**

114. MYSQL 서버 구축하기 2 - RDS 인스턴스 생성하기

**• 학습 내용:** AWS RDS 서버 인스턴스를 생성하는 방법을 이해한다.

**• 힌트 내용:** RDS 인스턴스 생성시 설정 사항을 확인한다.

AWS에 정상적으로 가입했다면, AWS 콘솔에 접속할 수 있다. AWS 웹 사이트(<https://aws.amazon.com/ko/>)에서 가입한 계정으로 로그인을 하고 [내 계정] - [AWS Management Console]을 클릭한다.

1. 왼쪽 상단의 [서비스] - [데이터베이스] - [RDS]를 클릭한다. 다음과 같이 RDS Dashboard 페이지에서 [데이터베이스 생성] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 154>**

2. 엔진 유형을 [MySQL]로 선택한다. 나머지 설정은 수정하지 않는다.

|  |
| --- |
| **<그림 155>** |

3. 템플릿을 [프리티어]로 선택한다. [DB 인스턴스 식별자]는 사용할 인스턴스의 명칭을 정해 입력한다. [마스터 암호]는 DB 툴(workbench)과 node 서버(mysql 연동 부분)에서 사용되므로 입력한 후 잊어버리지 않도록 한다. 나머지 설정은 수정하지 않는다.

|  |
| --- |
|  |

**<그림 156>**

4. [스토리지 자동 조정 활성화]는 사용량에 따라 과금될 수 있기 때문에 체크 표시를 해제한다. 나머지 설정은 수정하지 않는다.

|  |
| --- |
| **<그림 157>** |

5. [퍼블릭 엑세스 허용]을 [예]로 체크해 외부 DB 툴(workbench)에서도 접속할 수 있도록 한다. [데이터베이스 포트]가 3306인 것을 확인한다. 나머지 설정은 수정하지 않는다.

|  |
| --- |
| **<그림 158>** |

6. 마지막으로 [데이터베이스 생성] 버튼을 누르면, RDS 인스턴스가 생성된다.

|  |
| --- |
|  |

**<그림 159>**

7. 인스턴스 생성이 완료되면 다음과 같이 인스턴스 상태가 [사용 가능]으로 표시된다.

|  |
| --- |
|  |

**<그림 160>**

115. MYSQL 서버 구축하기 2 - RDS 인스턴스 포트 설정하기

**• 학습 내용:** AWS RDS 서버 인스턴스의 인바운드 포트 생성 방법을 이해한다.

**• 힌트 내용:** 인바운드 포트 설정을 하면, AWS 외부에서 RDS 서버에 접속할 수 있다.

AWS 인바운드 규칙을 추가하면, 특정 포트 트래픽의 접속을 허용할 수 있다.

1. 왼쪽 상단의 [서비스] - [컴퓨팅] - [EC2]를 클릭한다. 페이지 왼쪽에 표시되는 [보안 그룹] 버튼을 누르면, 다음과 같이 보안 그룹 리스트가 표시된다. RDS 인스턴스에서 사용 중인 [보안 그룹 ID]를 클릭한다. 별도의 설정이 없었다면 [보안 그룹 이름]이 default인 [보안 그룹 ID]를 선택한다.

|  |
| --- |
|  |

**<그림 161>**

2. 인바운트 포트를 추가하기 위해 [인바운드 규칙 편집] 버튼을 클릭한다.

|  |
| --- |
|  |

**<그림 162>**

3. [규칙 추가]를 누르고 다음과 같이 유형이 [MYSQL/Aurora]인 3306포트의 규칙을 입력하고 [규칙 저장]을 클릭한다.

|  |
| --- |
|  |

**<그림 163>**

116. MYSQL client 설치하기

**• 학습 내용:** mysql client 설치 방법을 이해한다.

**• 힌트 내용:** mysql installer를 실행하면서 workbench를 선택해 함께 설치한다.

mysql client는 로컬에 db 서버를 설치하지 않고 rds와 같은 외부 db 서버에 접속해서 사용할 경우에 설치하는 타입이다. client 타입은 로컬에 서버를 설치하는 server 타입보다 설치 용량이 작다. mysql installer를 실행하면서 db 툴로 사용할 workbench를 함께 설치할 수 있다.

1. mysql 다운로드 경로(<https://dev.mysql.com/downloads/installer/>)에 접속해 두 번째 파일(mysql-installer-community-8.0.20.0.msi) 오른쪽의 [Download] 버튼을 누른다. 버전은 예제와 다를 수 있는데, 설치 시점에 표시되는 최신 버전으로 다운로드한다.

|  |
| --- |
|  |

**<그림 164>**

2. 다음 페이지에서 [NO thanks, just start my download.] 버튼을 누르면 설치 파일이 다운로드된다.

|  |
| --- |
|  |

**<그림 165>**

3. 다운로드가 완료되면 installer를 실행한다. 설치 타입을 [Client only]로 선택하고 [Next] 버튼을 눌러 설치를 진행한다.

|  |
| --- |
|  |

**<그림 166>**

4. 설치 진행 도중 [Check Requirements] 단계에서 [MySQL Workbench 8.0.20]을 클릭한 후 [Execute]를 누르면 다음과 같이 설치가 시작된다. 마지막 단계까지 [Next]를 눌러 설치를 완료한다.

|  |
| --- |
|  |

**<그림 167>**

117. workbench로 rds 서버 접속하기

**• 학습 내용:** workbench를 실행해 RDS 인스턴스에 접속한다.

**• 힌트 내용:** rds 서버 주소와 계정 정보를 입력하면 workbench에서 접속할 수 있다.

mysql workbench는 오라클에서 개발한 데이터베이스 설계 도구다. workbench를 이용하면 스키마와 테이블 생성, 데이터 조회, 수정, 삭제, 삽입 등을 할 수 있다.

1. workbench에 입력할 rds 서버 주소를 확인한다. aws 로그인 후 콘솔에 접속한다. 왼쪽 상단의 [서비스] - [데이터베이스] - [RDS]를 클릭한다. Dashboard에서 [DB 인스턴스 (1/40)] 버튼을 클릭한다. [DB 식별자]를 클릭하면 다음과 같이 엔드포인트를 확인할 수 있는 페이지가 표시된다. 엔드포인트는 보안상의 이유로 생성된 rds 서버 주소다.

|  |
| --- |
|  |

**<그림 168>**

2. workbench를 실행한 후 [MySQL Connections] 문구 옆의 [+] 버튼을 누르면 [Setup New Connection] 창이 표시된다. [Connection Name]을 정해 입력한다. [Hostname]에 rds 서버 엔드포인트를 입력한다. [Username]은 rds 기본 설정인 admin으로 입력한다. [Store in Vault …] 버튼을 누르면 패스워드를 입력할 수 있다. RDS 인스턴스 생성 시에 작성한 마스터 암호를 입력한다. [OK]를 눌러 정상적으로 접속되는지 확인한다.

|  |
| --- |
|  |

**<그림 169>**

3. rds 서버에 정상적으로 접속됐다면, 다음과 같이 connection이 생성된다.

|  |
| --- |
|  |

**<그림 170>**

4. 생성된 connection 영역을 클릭하면, rds 서버에 접속되면서 다음과 같이 빈 스크립트가 표시된다.

|  |
| --- |
|  |

**<그림 171>**

118. MYSQL 스키마 생성 및 table 생성하기

**• 학습 내용:** workbench를 실행해 스키마와 table을 생성한다.

**• 힌트 내용:** 생성한 스키마에 table을 추가할 수 있다.

예제에서 사용하는 workbench 스키마는 데이터베이스 이론의 스키마(데이터베이스의 구조와 제약 조건을 기술한 것)와는 다른 용어다. 하나의 mysql 서버에서 여러 개의 데이터베이스 생성해 독립적으로 사용할 수 있는데, 이를 ‘스키마’라고 한다. 예를 들어, 스키마 A와 B를 만들고 두 스키마에 각각 동일한 테이블 C를 생성했다면, A 스키마의 C 테이블과 B 스키마의 C 테이블은 완전히 다른 테이블이다.

1. workbench에 접속한다. 왼쪽 [Navigator] 영역 하단의 [Schemas] 탭을 선택한 후 빈 공간에 마우스 오른쪽 버튼을 누르면 나타나는 단축 메뉴 중에서 [Create Schema…]를 클릭한다.

|  |
| --- |
|  |

**<그림 172>**

2. 다음과 같이 Schema 창이 나타나면, [Name]에 react라는 스키마명을 입력하고 [Apply] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 173>**

3. 다음 화면에서 [Apply] 버튼을 누르면, workbench가 [CREATE SCHEMA ‘react’] 명령어를 실행해 스키마를 생성한다.

|  |
| --- |
|  |

**<그림 174>**

4. 스키마 생성이 완료되면, 왼쪽 SCHEMAS 영역에 react라는 스키마가 표시된다. 스크립트 창에서 [use react] 쿼리를 실행해 react라는 스키마를 사용하겠다고 선언한다. 스키마를 선택한 후 테이블을 CREACT TABLE 쿼리로 생성한다. 쿼리 실행 방법은 실행하려는 쿼리를 블록을 지정하거나 세미콜론을 기준으로 실행하려는 쿼리 영역에 마우스 커서를 올려놓고 [Ctrl] + [Enter] 를 누르면 된다. 실행한 쿼리가 정상적으로 완료됐다면, Output 영역에 완료 표시가 표시된다.

|  |
| --- |
|  |

**<그림 175>**

119. MYSQL 한국 시간 설정하기

**• 학습 내용:** rds 서버 시간을 한국 시간으로 변경한다.

**• 힌트 내용:** rds 콘솔을 사용해 시간 설정을 변경한다.

AWS 서버가 한국에 있는 것이 아니기 때문에 초기 시간 값이 한국이 아닐 수 있다. db 서버의 시간은 데이터를 insert나 update할 때 기록하기 때문에 rds 서버 시간을 한국 시간에 맞게 변경해야 한다.

1. workbench에서 now() 함수로 현재 시간을 조회해보면, 한국 시간과 다르다는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 176>**

2. rds에 콘솔에 접속해 [ 파라미터 그룹 생성]을 누른다.

|  |
| --- |
|  |

**<그림 177>**

3. 그룹 이름을 입력한 후 [생성] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 178>**

4. 생성된 파라미터명을 클릭해 수정 페이지로 진입한다.

|  |
| --- |
|  |

**<그림 179>**

5. 검색 창에 [time\_zone]을 검색해 [Asia/Seoul]을 선택한 후 [변경 사항 저장]을 누른다.

|  |
| --- |
|  |

**<그림 180>**

6. RDS 인스턴스에 추가한 파라미터 그룹을 등록하기 위해 [수정] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 181>**

7. [DB 파라미터 그룹]에 새로 추가한 파라미터 그룹을 선택한 후 [계속] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 182>**

8. 다음 페이지에서 [즉시 적용]을 선택한 후 [DB 인스턴스 수정] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 183>**

9. RDS 인스턴스를 재부팅한다.

|  |
| --- |
|  |

**<그림 184>**

10. 다시 workbench에서 now() 함수를 실행하면, 한국 시간이 조회되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 185>**

120. MYSQL 한글 설정 및 Safe 모드 해제

**• 학습 내용:** rds 서버에 한글 설정과 safe 모드를 해제하는 방법을 이해한다.

**• 힌트 내용:** 생성한 테이블에 데이터를 삽입, 수정, 삭제하기 위한 설정을 한다.

rds 서버에 한글 설정이 돼 있지 않다면, 데이터를 삽입할 때 한글이 깨지게 된다. 테이블별로 charset을 utf8로 변경하면 한글 깨짐 현상을 해결할 수 있다. safe 모드가 설정돼 있으면 update와 delete 쿼리를 사용할 수 없기 때문에 해제해야 한다.

1. workbench에서 SQL\_SAFE\_UPDATES 값을 0으로 세팅하면 safe 모드가 해제되고 delete문이 실행되는 것을 확인할 수 있다. ALTER TABLE 쿼리를 이용해 특정 테이블의 charset을 utf8로 수정하면, 한글이 깨지지 않도록 변경할 수 있다.

|  |
| --- |
|  |

**<그림 186>**

121. MYSQL 더미 데이터 삽입하기

**• 학습 내용:** react 페이지에서 표시할 더미 데이터를 삽입한다.

**• 힌트 내용:** worckbench에서 insert문을 사용해 데이터를 삽입한다.

react에서 등록 페이지를 만들기 전에 리스트 조회 페이지를 구현한다면, 화면에 표시할 데이터가 없을 것이다. 이런 상황에 테스트용으로 삽입하는 유용하지 않은 데이터를 ‘더미 데이터’라 한다.

1. 다음과 같이 insert문을 사용해 더미 데이터를 삽입하고 select문으로 테이블을 조회하면 삽입된 데이터를 확인할 수 있다.

|  |
| --- |
|  |

**<그림 187>**

122. NODE 조회 Api 만들기 1 - body-parser 패키지 사용하기

**• 학습 내용:** body-parser 패키지를 사용해 http request에 담긴 데이터를 불러온다.

**• 힌트 내용:** Api 호출 url에 붙은 파라미터나 http body에 담긴 데이터를 사용할 수 있다.

post 방식으로 api를 호출하는 경우, 전달할 데이터들을 request body에 담아 전달할 수 있다. 호출받은 node 서버에서 body-parser 패키지를 사용하면, 간편하게 body에 있는 파리미터를 추출해 사용할 수 있다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save body-parser]를 입력하면 다음과 같이 npm이 body-parser를 설치한다.

|  |
| --- |
|  |

**<그림 188>**

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7 | .. 코드 생략..  var swtoolRouter = require("./routes/SwtoolRout");  .. 코드 생략..  app.use("/api/Swtool", swtoolRouter);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

4: /api/Swtool 경로로 api가 호출되면, line 2에서 지정한 파일 경로로 라우팅한다.

routes 폴더의 SwtoolRout.js 파일을 확인한다.

<소스> SwtoolRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30 | var express = require('express');  var router = express.Router();  const bodyParser = require('body-parser');  router.use(bodyParser.urlencoded({ extended: true }));  router.post('/', (req, res, next) => {  var type = req.query.type;  if(type == 'list'){  //Swtool 리스트 조회  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');    //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'SwToolsMapper';//mybatis xml 파일명  req.body.crud = 'select';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'selectSwToolsList';    router.use('/', dbconnect\_Module);  next('route')  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  });  module.exports = router; |

4: body-parser 패키지를 사용할 수 있도록 require한다.

6: body-parser 패키지의 urlencoded 함수를 실행하면, &key1=value1&key2=value2와 같은 형태로 전달되는 데이터를 추출할 수 있다.

10~11: <http://localhost:5000/api/Swtool?type=list>와 같은 호출 url에서 type이라는 key의 value(list)를 추출하기 위해서는 req.query 문법으로 key 값에 접근해야 한다. type 값에 따라 호출하는 쿼리 정보를 분기 처리한다.

18~20: 쿼리가 작성될 xml 파일명(mapper), 작업 종류(crud), 실행 쿼리의 id(mapper\_id) 정보를 resquest body에 넣는다. json 형태의 데이터로 할당된다.

22: line 15에서 require한 dbconnect\_Module 파일로 라우팅한다. dbconnect\_Module 파일도 라우터 역할을 하는데, mysql 서버를 연결해 쿼리를 실행하고 결괏값을 response로 전송한다. 이때 line 18~20에서 body에 넣었던 데이터들이 함께 전송된다.

23: next 함수를 사용하면 현재 라우터에서 response를 보내지 않고 다음 라우터(dbconnect\_Module)로 response 작업을 넘길 수 있다.

123. NODE 조회 Api 만들기 2 - mysql 서버 연결하기

**• 학습 내용:** node 서버와 mysql 서버를 연결해 데이터를 불러온다.

**• 힌트 내용:** workbench에서 사용하던 rds 주소와 계정 정보로, mysql 서버를 연결할 수 있다.

mysql 패키지를 설치하면 node 서버와 mysql 서버를 연결해 select, insert, update, delete와 같은 쿼리를 호출할 수 있다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save mysql]를 입력하면 다음과 같이 npm이 mysql를 설치한다.

|  |
| --- |
|  |

**<그림 189>**

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27 | const mysql = require("mysql");  .. 코드 생략..  //mysql 서버 접속 정보  const connection = mysql.createConnection({  host: "react200.#########.ap-northeast-2.rds.amazonaws.com",  port: "3306",  database: 'react',  user: "admin",  password: "react200RDS",  });  router.post("/", (req, res) => {  .. 코드 생략..  connection.query(query, function (error, results) {  .. 코드 생략..  string = JSON.stringify(results);  var json = JSON.parse(string);  res.send({ json });  console.log("========= Node Mybatis Query Log End =========\n");  });  });  module.exports = router; |

1: mysql 패키지를 require해 사용할 수 있도록 한다.

5~12: mysql 서버 연결 정보를 사용해 createConnection 함수로 mysql 서버를 연결한다. 생성된 연결(connection)을 connection 변수에 할당한다. host에 rds 엔드포인트 주소를 넣는다. port는 rds 서버에서 사용하는 3306을 입력한다. database는 workbench에서 추가한 스키마명을 할당한다. user와 password는 workbench에서 사용하던 계정과 비밀번호를 그대로 사용한다.

16: 쿼리 정보가 담긴 query 변수의 데이터를 query 함수의 파라미터로 넣고 실행한다. query 변수에 데이터를 할당하는 부분은 mybatis-mapper 패키지가 사용되기 때문에 생략됐고 다음 예제에서 확인하도록 한다.

20: 쿼리를 실행한 후 결괏값을 response(res)에 담아 send 함수로 전송한다. Line 18~19은 json 데이터를 string으로 바꾸고 다시 json으로 바꾸는 코드인데, react에서 사용할 수 있는 json 데이터로 변환하기 위해 실행한다.

124. NODE 조회 Api 만들기 3 - mybatis 패키지 사용하기

**• 학습 내용:** java 언어에서 사용하는 mybatis라는 프레임워크를 이해한다.

**• 힌트 내용:** mybatis의 장점과 node에서 사용하는 방법을 확인한다.

mybatis는 java 언어에서 데이터베이스를 연결할 때 사용하는 프레임워크다. mybatis가 없어도 데이터베이스를 연결할 수 있다. mybatis를 사용하면 java 코드에서 sql 코드를 분리해 별도의 파일로 관리할 수 있어 코드 재사용성이 높아진다. node에서도 이와 같은 이유로 mybatis를 사용한다. 또 workbench와 같은 데이터베이스 툴에서 사용하던 쿼리를 그대로 복사해 사용할 수 있으므로 개발의 편의성을 높여준다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save mybatis-mapper]를 입력하면 다음과 같이 npm이 mybatis-mapper를 설치한다.

|  |
| --- |
|  |

**<그림 190>**

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49 | var express = require("express");  var router = express.Router();  const mysql = require("mysql");  const bodyParser = require("body-parser");  router.use(bodyParser.json());  //mysql 서버 접속 정보  const connection = mysql.createConnection({  host: "react200.#########.ap-northeast-2.rds.amazonaws.com",  port: "3306",  database: 'react',  user: "admin",  password: "react200RDS",  });  router.post("/", (req, res) => {  const mybatisMapper = require("mybatis-mapper");  var param = req.body;  //mybatis mapper 경로 설정  mybatisMapper.createMapper(['./models/'+param.mapper+'.xml']);  var time = new Date();  console.log('## '+time+ ' ##');  console.log("\n Called Mapper Name = "+param.mapper);  var format = { language: 'sql', indent: ' ' };  //mysql 쿼리 정보 세팅  var query = mybatisMapper.getStatement(param.mapper, param.mapper\_id, param, format);  console.log("\n========= Node Mybatis Query Log Start =========");  console.log("\* mapper namespce : "+param.mapper+"."+param.mapper\_id+" \*\n");  console.log(query+"\n");  connection.query(query, function (error, results) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  string = JSON.stringify(results);  var json = JSON.parse(string);  res.send({ json });  console.log("========= Node Mybatis Query Log End =========\n");  });  });  module.exports = router; |

19: mybatis-mapper 패키지를 require해 사용할 수 있도록 한다.

20: SwtoolRout.js 코드에서 req.body에 json 형태의 데이터를 할당했다. body에 있는 json 데이터를 가져오기 위해 line 6에 bodyParser.json() 함수를 사용한다.

23: mybatis-mapper 패키지의 createMapper 함수를 사용해 쿼리를 작성할 xml 파일 경로를 파라미터로 전달한다.

24~26: 쿼리가 실행되는 시간과 xml 파일명을 로그로 출력한다.

28: mybatis에서 mysql을 사용하기 때문에 format의 언어를 sql로 할당한다.

30: mysql에서 실행할 쿼리를 getStatement가 생성해 query라는 변수에 할당한다. getStatement 함수의 파라미터로 쿼리가 작성될 xml 파일(mapper)명, xml 파일 안에서 실행될 특정 쿼리의 id(mapper\_id), req.body로 전달된 파라미터(param) 그리고 format 변수 순서로 세팅한다. 이때 세 번째 파라미터인 req.body 데이터에는 현재 쿼리에서 사용할 데이터는 없다. 하지만 where절이 있는 select문을 사용하거나 insert문, update문에 필요한 모든 데이터를 req.body에 담아 쿼리로 전달하게 된다.

32: xml 파일명과 실행 쿼리의 id를 로그로 출력한다.

33: 실행 쿼리를 로그로 출력한다. 로그에 출력된 쿼리를 수정할 필요 없이 그대로 복사해 workbench에서 실행해 결과를 확인할 수 있다.

35: 생성된 연결(connection)에 query 함수를 사용해 query 변수에 저장된 쿼리를 불러와 실행한다. mysql 서버에서 실행 결과를 반환하면, 콜백 함수의 파라미터인 results 변수에 할당된다.

41: 쿼리 실행 결과를 로그로 출력한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="SwToolsMapper">  <select id="selectSwToolsList">  SELECT  swt\_code  , swt\_toolname  , swt\_function  , swt\_imagepath  , swt\_big\_imgpath  , swt\_comments  , swt\_demo\_site  , swt\_manual\_path  , swt\_github\_url  , reg\_date  FROM react.react\_swtool  ORDER BY update\_date DESC  </select>  </mapper> |

4: mapper의 namespace에 xml 파일명을 입력한다. namespace는 mybatis 내에서 원하는 sql 쿼리를 찾아 실행하기 위해 사용하는 변수다.

5: 실행할 쿼리의 종류(select, insert, update, delete)과 mapper의 id를 입력한다. mapper id는 하나의 xml 파일 안에서 유일한 값이어야 한다.

6~19: 실행할 쿼리를 입력한다. 이 쿼리를 그대로 workbench에서 실행할 수 있다.

125. NODE 조회 Api 만들기 4 - Api 호출로 데이터 확인하기

**• 학습 내용:** Api 호출 툴을 사용해 post 방식으로 node api를 호출한다.

**• 힌트 내용:** 실행 쿼리와 실행 결과가 출력된 로그를 확인한다.

크롬 웹 스토어에서 [Advanced REST client]라는 확장 프로그램을 설치한다.

|  |
| --- |
|  |

**<그림 191>**

[Advanced REST client] 프로그램을 실행한다. 다음과 같이 node api 주소를 post 방식으로 호출하면 json 형태의 데이터를 확인할 수 있다.

|  |
| --- |
|  |

**<그림 192>**

api를 호출할 때 cmd 창을 확인하면, 다음과 같이 실행 쿼리와 실행 결과가 로그로 출력된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 193>**

126. REACT 조회 페이지 만들기

**• 학습 내용:** react에서 node api를 호출해 데이터를 가져와 리스트에 표시한다.

**• 힌트 내용:** api 호출 결과 데이터를 가공하고 반복문으로 사용해 리스트를 만든다.

react 컴포넌트에서 post 방식으로 axios를 사용해 node api를 호출한다. 호출된 데이터를 가공해 html 코드로 만들고 state 변수에 할당한다. html 코드가 할당된 state 변수를 render 함수를 통해 화면에 표시한다.

SoftwareToolsManage 폴더의 SoftwareList.js 파일을 확인한다.

<소스> SoftwareList.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49  50  51  52  53 54  55  56  57  58  59  60  61  62 63  64  65  66  67  68  69  70  71 72  73  74 | .. 코드 생략..  componentDidMount() {  this.callSwToolListApi()  }  callSwToolListApi = async () => {  axios.post('/api/Swtool?type=list', {  })  .then( response => {  try {  this.setState({ responseSwtoolList: response });  this.setState({ append\_SwtoolList: this.SwToolListAppend() });  } catch (error) {  alert('작업 중 에러가 발생했습니다.');  }  })  .catch( error => {alert('작업 중 에러가 발생했습니다.');return false;} );  }  SwToolListAppend = () => {  let result = []  var SwToolList = this.state.responseSwtoolList.data    for(let i=0; i<SwToolList.json.length; i++){  var data = SwToolList.json[i]  var date = data.reg\_date  var year = date.substr(0,4)  var month = date.substr(4,2)  var day = date.substr(6,2)  var reg\_date = year +'.'+month+'.'+day  result.push(  <tr class="hidden\_type">  <td>{data.swt\_toolname}</td>  <td>{data.swt\_function}</td>  <td>{reg\_date}</td>  <td>  <Link to={'/AdminSoftwareView/'+data.swt\_code}  className="bt\_c1 bt\_c2 w50\_b">수정</Link>  <a href="#n" class="bt\_c1 w50\_b" >삭제</a>  </td>  </tr>  )  }  return result  }  render () {  return (  <section class="sub\_wrap" >  <article class="s\_cnt mp\_pro\_li ct1 mp\_pro\_li\_admin">  .. 코드 생략..  <div class="list\_cont list\_cont\_admin">  <table class="table\_ty1 ad\_tlist">  <tr>  <th>툴 이름</th>  <th> 기능</th>  <th>등록일</th>  <th> 기능</th>  </tr>  </table>  <table class="table\_ty2 ad\_tlist">  {this.state.append\_SwtoolList}  </table>  </div>  </article>  </section>  );  }  }  export default SoftwareList; |

50~71: 생명주기 실행 함수 순서에 따라 render 함수가 line 2의 componentDidMount 함수보다 먼저 실행된다. Line 56~63의 <table> 태그로 리스트의 헤더를 화면에 나타낸다. Line 64~66의 <table> 태그로 본문의 리스트 내용을 나타낸다. state 변수 append\_SwtoolList는 rend 함수가 처음 실행되는 시점에 빈 값이기 때문에 화면에 빈 리스트가 표시된다.

2~4: render 함수가 실행되고 componentDidMount 함수가 실행된다. componentDidMount 함수 안에서 node api를 호출하는 callSwToolListApi 함수를 호출한다.

6~19: axios로 node api 경로(/api/Swtool?type=list)를 호출한다. api 호출 결과(response)를 state 변수 responseSwtoolList:에 할당한다. 배열을 반환하는 SwToolListAppend 함수를 실행하고 return 값을 state 변수 append\_SwtoolList에 할당한다.

23: SwToolListAppend 함수가 실행되면, line 12에서 할당한 state 변수 responseSwtoolList에 .data로 접근해 지역 변수 SwToolList 변수에 할당한다.

|  |
| --- |
| NOTE |
| json 데이터는 마침표(.)를 통해 하위 데이터로 접근할 수 있는데, 최종적으로 사용할 데이터는 this.state.responseSwtoolList.data.json[i]. 변수명 형태다. 코드의 사용량을 줄이기 위해 SwToolList 변수에 할당해 사용한다. |

25: 사용할 json 데이터 배열의 크기를 length 함수를 통해 가져오고 0부터 배열의 크기보다 하나 작은 값까지 i 변수를 증가시키면서 반복문을 실행한다.

26: json 배열의 첫 번째 데이터부터 순서대로 data라는 지역 변수에 할당한다.

28~32: 날짜 데이터의 형태를 변경하기 위해 substr 함수를 사용한다.

34~45: for문으로 반복문을 한 번 실행할 때마다 <tr> 태그(table의 행)을 생성해 result라는 배열에 추가한다. result는 line 22에서 배열로 선언했는데, push 함수를 사용해 순서대로 데이터를 삽입할 수 있다.

36~43: 행 안에 가로 열을 구성할 <td> 태그를 추가한다. 1열당 4개의 가로 열을 추가할 것이기 때문에 <td> 태그를 4번 작성한다. data.변수명으로 사용할 데이터를 가져올 수 있는데, 이때 변수명은 SwToolsMapper.xml 파일에서 select한 react\_swtool 테이블의 컬럼명과 동일하다.

47: 반복문이 완료되면 json 데이터의 크기만큼의 <tr> 태그가 result 배열에 할당된다. result 변수를 return한다.

13: return된 result 변수의 값을 state 변수 append\_SwtoolList에 할당한다. setState 함수를 사용하면 state 변수에 값이 할당되고 render 함수를 다시 실행한다.

65: render 함수가 재실행되면서 append\_SwtoolList에 할당됐던 html 코드(<tr> 태그 리스트)가 화면에 표시된다.

SoftwareList 컴포넌트의 결과를 확인하기 위해 App.js에서 라우팅돼 있는 /SoftwareList 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | .. 코드 생략..  import SoftwareList from './SoftwareToolsManage/SoftwareList';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Route path='/SoftwareList' component={SoftwareList} />  .. 코드 생략.. |

|  |
| --- |
|  |

**<그림 194>**

127. DB 연결 풀 구현하기

**• 학습 내용:** mysql 패키지에서 지원하는 connection pool을 구현한다.

**• 힌트 내용:** DB를 효율적으로 연결하기 위해 connection pool 사용 방법을 확인한다.

사용자가 웹 사이트에 접속해 db를 사용하게 되면, node 서버에서 mysql 서버로 connection을 만든다. 그리고 db 사용이 완료되면 connection을 종료한다. 이때 node 서버와 mysql 서버 사이에 connection을 만드는 작업은 시간적 자원이 소모된다. connection pool은 한 번 맺은 connection을 종료시키지 않고 pool에 저장한다. 그리고 새로운 connection 요청이 오면, pool에 저장된 connection을 제공한다. 이렇게 connection pool을 사용하면 새로운 connection을 만드는데 필요한 자원 사용을 감소시킬 수 있다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25 | .. 코드 생략..  // Connection Pool 세팅  const pool = mysql.createPool({  connectionLimit: 66,  waitForConnections: true,  host: "react200.#########.ap-northeast-2.rds.amazonaws.com",  port: "3306",  database: 'react',  user: "admin",  password: "react200RDS",  });  .. 코드 생략..  pool.getConnection(function(err,connection){  connection.query(query, function (error, results) {  .. 코드 생략..  var json = JSON.parse(string);  res.send({ json });  connection.release();  console.log("========= Node Mybatis Query Log End =========\n");  });  })  });  module.exports = router; |

3: mysql 패키지의 createConnection 함수 대신 createPool 함수를 사용해 connection pool을 생성한다.

4~5: connectionLimit 옵션 값은 connection pool에 최대로 저장할 수 있는 connection의 수다. 예제에서 사용하는 rds 서버(t2.micro)가 지원하는 최대 연결 수는 66개다. waitForConnections 옵션은 pool의 connection이 모두 사용 중일 때 새로운 connection 요청이 들어온 경우 대기 여부를 결정한다. 값이 true인 경우 여유 connection이 생길 때까지 대기시킨다.

14~15: pool에 getConnection 함수를 실행해 연결을 생성한다. 생성된 connection은 콜백 함수를 통해 connection 변수로 반환되고 반환된 connection에 query 함수를 사용해 쿼리를 실행한다.

실행 결과는 connection pool을 사용할 때와 사용하지 않았을 때가 동일하다.

|  |
| --- |
|  |

**<그림 195>**

128. NODE 등록 Api 만들기 1 - insert 라우터 분기

**• 학습 내용:** 데이터를 삽입하기 위한 post 방식의 api를 구현한다.

**• 힌트 내용:** node 라우터 코드에서 사용할 mapper 정보를 분기 처리한다.

조회 api를 구현할 때 데이터 연동을 위한 기본 틀이 갖춰졌다. 전달하는 데이터만 달라질 뿐 등록 수정, 삭제 기능도 동일한 틀 안에서 분기 처리해 사용할 수 있다. react에서 호출할 등록 api는 /api/Swtool?type=save이다. ? 뒤에 파라미터 type 변숫값만 다를 뿐, 호출 url은 동일하다. 그렇기 때문에 server.js 파일은 수정사항이 없다.

routes 폴더의 SwtoolRout.js 파일을 확인한다.

<소스> SwtoolRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25 | .. 코드 생략..  router.post('/', (req, res, next) => {  var type = req.query.type;  if(type == 'list'){  .. 코드 생략..  }else if(type == 'save'){  //Swtool 관리자 저장  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');    //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'SwToolsMapper';//mybatis xml 파일명  req.body.crud = 'insert';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'insertSwToolsInfo';    router.use('/', dbconnect\_Module);  next('route')  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  });  .. 코드 생략.. |

14~16: 파라미터로 전달받은 type 값이 save인 경우, insert 쿼리를 실행할 mapper 정보를 body에 할당한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35 | .. 코드 생략..  router.post("/", (req, res) => {  const mybatisMapper = require("mybatis-mapper");  var param = req.body;  //mybatis mapper 경로 설정  mybatisMapper.createMapper(['./models/'+param.mapper+'.xml']);  var time = new Date();  console.log('## '+time+ ' ##');  console.log("\n Called Mapper Name = "+param.mapper);  var format = { language: 'sql', indent: ' ' };  //mysql 쿼리 정보 세팅  var query = mybatisMapper.getStatement(param.mapper, param.mapper\_id, param, format);  console.log("\n========= Node Mybatis Query Log Start =========");  console.log("\* mapper namespce : "+param.mapper+"."+param.mapper\_id+" \*\n");  console.log(query+"\n");  pool.getConnection(function(err,connection){  connection.query(query, function (error, results) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  .. 코드 생략..  connection.release();  console.log("========= Node Mybatis Query Log End =========\n");  });  })  });  module.exports = router; |

dbconnect\_Module.js 파일은 mysql 서버(rds) 접속 정보, mybatis 연동, connection pool에 대한 코드가 구현돼 있다. dbconnect\_Module.js은 모듈화해 공통 코드로 사용하고 SwtoolRout.js 라우터 파일에서 전달받은 파라미터에 따라 mapper 정보를 불러와 쿼리를 실행하고 react 페이지로 response를 전달한다.

8: SwtoolRout.js에서 전달받은 req.body.mapper 값인 SwToolsMapper로 사용할 mapper 경로를 설정한다.

15: SwtoolRout.js에서 전달받은 req.body.mapper\_id 값인 insertSwToolsInfo로 query 변수를 생성한다.

129. NODE 등록 Api 만들기 2 - insert 쿼리 추가, response 처리

**• 학습 내용:** mapper 파일에 insert 쿼리를 추가한 후 실행 결과에 따라 response를 보낸다.

**• 힌트 내용:** select와 달리 insert는 react 페이지로 전송할 데이터가 없다.

xml 파일(mapper)에 쿼리를 추가하는 방식은 select 대신 <insert> 태그를 사용하는 것 외에는 select에서 사용한 것과 같다. 쿼리를 실행한 후 select는 조회된 데이터를 response로 넘기지만, insert는 성공 여부만 전달한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49  50  51  52 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="SwToolsMapper">  .. 코드 생략..  <insert id="insertSwToolsInfo">  INSERT INTO react.react\_swtool  (  swt\_code  , swt\_toolname  , swt\_function  , swt\_comments  , swt\_demo\_site  , swt\_github\_url  <if test="is\_LabelImg != null && is\_LabelImg != ''">  , swt\_imagepath  </if>  <if test="is\_MainImg != null && is\_MainImg != ''">  , swt\_big\_imgpath  </if>  <if test="is\_MenualName != null && is\_MenualName != ''">  , swt\_manual\_path  </if>  , reg\_date  , reg\_user  , update\_date  , update\_user  )  VALUES (  CONCAT('USW', DATE\_FORMAT(now(), '%Y%m%d%H%i%s'))  , #{is\_Swt\_toolname}  , #{is\_Swt\_function}  , #{is\_Comments}  , #{is\_Swt\_demo\_site}  , #{is\_Giturl}  <if test="is\_LabelImg != null && is\_LabelImg != ''">  , #{is\_LabelImg}  </if>  <if test="is\_MainImg != null && is\_MainImg != ''">  , #{is\_MainImg}  </if>  <if test="is\_MenualName != null && is\_MenualName != ''">  , #{is\_MenualName}  </if>  , DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , #{is\_Email}  , DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , #{is\_Email}  )  </insert>  </mapper> |

6: <insert> 태그를 사용해 mapperid를 정의한다.

8~50: 실행할 insert 쿼리를 작성한다.

16~18: mybatis에서는 전달받은 파라미터로 if문을 사용할 수 있다. 필수 값이 아닌 컬럼 값이 null이나 공백으로 넘어올 경우에는 테이블에 삽입하지 않는다.

31: 테이블에서 유일한 값을 갖는 기본 키인 swt\_code의 값은 USW라는 문자열에 현재 시간 정보를 붙여 생성한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | .. 코드 생략..  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  if(results != undefined){  string = JSON.stringify(results);  var json = JSON.parse(string);  if (req.body.crud == "select") {  res.send({ json });  }else{  res.send("succ");  }  }else{  res.send("error");  }  .. 코드 생략.. |

9~13: SwtoolRout.js에서 할당한 req.body.crud 값에 따라 response로 전송할 데이터를 분기 처리한다. line 10 조회(select)인 경우 db에서 조회한 데이터를 json 형태로 response에 담아 react 페이지로 다시 보내야 한다. line 12는 등록(insert), 수정(update), 삭제(delete)인 경우에 해당하는데, 쿼리 실행이 성공했을 경우 succ라는 문자열을 전송한다.

14~16: 쿼리 실행 결과가 undefined라면 쿼리가 정상적으로 실행되지 않은 경우다. 이때는 error라는 문자열을 전송한다. react 페이지에서는 response에 전송된 문자열에 따라 사용자에게 적절한 메시지를 표시할 수 있다.

130. REACT 등록 페이지 만들기

**• 학습 내용:** react에서 node api를 호출해 전달한 데이터를 table에 삽입한다.

**• 힌트 내용:** api 호출 결과에 따라 사용자에게 적절한 메시지를 표시한다.

react 컴포넌트에서 post 방식으로 axios를 사용해 node api를 호출한다. Insert 쿼리에 사용할 데이터를 json 형태로 변경해 전달하고 쿼리가 정상적으로 실행됐다면 완료 후 리스트 페이지로 이동한다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49  50  51  52  53 54  55  56  57  58  59  60  61  62 63  64  65  66  67  68  69  70  71 72  73  74  75  76  77  78 79  80  81  82  83  84  85 86 | .. 코드 생략..  class SoftwareView extends Component {  submitClick = async (type, e) => {  this.Swt\_toolname\_checker = $('#is\_Swt\_toolname').val();  .. 코드 생략..  this.fnValidate = (e) => {  if(this.Swt\_toolname\_checker === '') {  $('#is\_Swt\_toolname').addClass('border\_validate\_err');  alert('툴 이름을 다시 확인해주세요.')  return false;  }  .. 코드 생략..  return true;  }  if(this.fnValidate()){  var jsonstr = $("form[name='frm']").serialize();  jsonstr = decodeURIComponent(jsonstr);  var Json\_form = JSON.stringify(jsonstr).replace(/\"/gi,'')  Json\_form = "{\"" +Json\_form.replace(/\&/g,'\",\"').replace(/=/gi,'\":"')+"\"}";    try {  const response = await fetch('/api/Swtool?type='+type, {  method: 'POST',  headers: {  'Content-Type': 'application/json',  },  body: Json\_form,  });  const body = await response.text();  if(body == "succ"){  if(type == 'save'){  this.sweetalertSucc('Software Tools 등록이 완료됐습니다.', false)  }  setTimeout(function() {  this.props.history.push('/SoftwareList');  }.bind(this),1500  );  }else{  alert('작업 중 에러가 발생했습니다.')  }  } catch (error) {  alert('작업 중 에러가 발생했습니다.')  }  }  };  .. 코드 생략..  render () {  return (  <section class="sub\_wrap">  <article class="s\_cnt mp\_pro\_li ct1">  <div class="li\_top">  <h2 class="s\_tit1">Software Tools 등록/수정</h2>  </div>  <div class="bo\_w re1\_wrap re1\_wrap\_writer">  <form name="frm" id="frm" action="" onsubmit="" method="post" >  <input id="is\_Swtcode" type="hidden" name="is\_Swtcode" />  <input id="is\_Email" type="hidden" name="is\_Email" value="guest" />  .. 코드 생략..  <table class="table\_ty1">  <tr>  <th>  <label for="is\_Swt\_toolname">툴 이름<span class="red">(\*)</span></label>  </th>  <td>  <input type="text" name="is\_Swt\_toolname" id="is\_Swt\_toolname" class="" />  </td>  </tr>  .. 코드 생략..  </table>  <div class="btn\_confirm mt20" style={{"margin-bottom": "44px"}}>  <Link to={'/SoftwareList'} className="bt\_ty bt\_ty1 cancel\_ty1">취소</Link>  <a href="javascript:" className="bt\_ty bt\_ty2 submit\_ty1 saveclass"  onClick={(e) => this.submitClick('save', e)}>저장</a>  .. 코드 생략..  </form>  </div>  </article>  </section>  );  }  }  export default SoftwareView; |

75~76: [저장] 버튼을 누르면 submitClick 함수를 호출하고 save 문자열을 파라미터로 보낸다.

5: submitClick 함수가 실행되면 입력한 값들을 id 값으로 접근하고 value 값을 가져와 지역 변수(Swt\_toolname\_checker)에 할당한다.

8~16: line 18에서 fnValidate 함수의 반환 값이 true인 경우, 등록 api를 호출한다. fnValidate 함수는 입력 값들이 공백일 경우, 사용자에게 메시지를 띄우고 false를 반환한다.

19~22: <form> 태그의 데이터를 수집해 json 형태로 가공한다. serialize 함수를 사용하면 <form> 태그로 감싸인(line 58~78)의 모든 <input> 태그의 name과 value를 한번에 가져올 수 있다.

25~31: fetch 함수를 사용해 node api(/api/Swtool?type=save)를 post 방식으로 호출한다. body에는 line 22에서 할당한 json 데이터를 담아 전달한다.

|  |
| --- |
| NOTE |
| <form> 태그를 사용하지 않는다면 line 30에서 body를 할당할 때 모든 <input> 태그의 name과 value 값을 하나씩 가져와 json 데이터를 생성해야 한다. 등록 페이지와 같이 입력 값이 많은 경우 <form> 태그를 사용하면 효율적으로 데이터를 전달할 수 있다. |

32: insert는 response 값으로 succ 또는 error 두 가지 종류의 문자열을 받기 때문에 text 함수를 사용해 값을 가져온다.

35: 정상적으로 insert문이 실행돼 succ가 반환됐다면 사용자에게 등록 완료 메시지를 표시한다.

38: 등록 완료 메시지를 표시한 후 리스트 페이지로 이동한다.

42: error가 반환된 경우 작업 중 에러 메시지를 표시한다.

SoftwareView 컴포넌트에서 등록 기능을 확인하기 위해 App.js에서 라우팅돼 있는 / SoftwareView 경로를 호출한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11  12 | .. 코드 생략..  import SoftwareView from './SoftwareToolsManage/SoftwareView';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Route path='/SoftwareList' component={SoftwareList} />  <Route path='/SoftwareView' component={SoftwareView} />  .. 코드 생략.. |

등록 페이지에서 다음과 같이 값들을 입력하고 [저장] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 196>**

데이터가 정상적으로 삽입되면, 리스트 페이지에서 추가된 데이터를 확인할 수 있다.

|  |
| --- |
|  |

**<그림 197>**

131. REACT 상세 조회 페이지 만들기

**• 학습 내용:** 등록된 데이터를 확인할 수 있는 상세 조회 페이지를 만든다.

**• 힌트 내용:** 상세 조회 페이지에서는 리스트 페이지에서 사용한 node api를 재사용할 수 있다.

리스트 페이지(<http://localhost:3000/SoftwareList>)에는 전체 데이터 목록이 출력되기 때문에 꼭 필요한 일부 데이터만 표시한다. 리스트에서 상세 데이터를 조회할 행의 [수정] 버튼을 누르면, 상세 조회 페이지로 이동하도록 구현할 수 있다.

SoftwareToolsManage 폴더의 SoftwareList.js 파일을 확인한다. 반복문을 사용해 리스트의 행(<tr> 태그)를 하나씩 생성하는 부분과 [Tools 등록] 버튼의 코드다.

<소스> SoftwareList.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22 | .. 코드 생략..  result.push(  <tr class="hidden\_type">  <td>{data.swt\_toolname}</td>  <td>{data.swt\_function}</td>  <td>{reg\_date}</td>  <td>  <Link to={'/SoftwareView/'+data.swt\_code}  className="bt\_c1 bt\_c2 w50\_b">수정</Link>  <a href="#n" class="bt\_c1 w50\_b" >삭제</a>  </td>  </tr>  )  .. 코드 생략..  <div class="li\_top">  <h2 class="s\_tit1">Software Tools 목록</h2>  <div class="li\_top\_sch af">  <Link to={'/SoftwareView/register'} className="sch\_bt2 wi\_au">Tool 등록</Link>  </div>  </div> .. 코드 생략.. |

9: [수정] 버튼을 <Link> 태그로 감싸고 to 속성에 이동할 컴포넌트 경로를 입력한다. 수정 페이지의 경로(/SoftwareView)뒤에 /를 추가한 후 data.swt\_code 변숫값을 추가한다. swt\_code는 react\_swtool 테이블의 기본 키로 유일한 값을 가진다. swt\_code 값을 쿼리의 where절에서 조회하면, 상세 페이지에서 조회할 데이터를 가져오게 된다.

19: 등록 페이지와 수정 페이지 모두 같은 경로(/SoftwareView)를 사용한다. SoftwareView 컴포넌트를 등록 페이지로 사용할 때는 register라는 상수를 경로에 붙여 전달한다.

components 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9 10 | .. 코드 생략..  <div className="App">  <HeaderAdmin/>  <Route exact path='/' component={LoginForm} />  <Route path='/SoftwareList' component={SoftwareList} />  <Route path='/SoftwareView/:swtcode' component={SoftwareView} />  <Footer/>  </div>  .. 코드 생략.. |

6: <Route> 태그의 path 속성에 /: 변수명 형태로 추가하면, /SoftwareView/ 경로 뒤에 붙어오는 문자열을 변수에 저장할 수 있다. 예를 들면, 리스트 페이지에서 [Tools 등록] 버튼을 눌러 /SoftwareView/register를 호출했다면 swtcode의 값이 register가 된다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43 | .. 코드 생략..  class SoftwareView extends Component {  constructor(props) {  super(props);  this.state = {  before\_swtcode: props.match.params.swtcode  }  }  componentDidMount () {  if(this.state.before\_swtcode == 'register'){  $('.modifyclass').hide()  }else{  this.callSwToolInfoApi()  $('.saveclass').hide()  }  }  callSwToolInfoApi = async () => {  axios.post('/api/Swtool?type=list', {  is\_Swtcode: this.state.before\_swtcode,  })  .then( response => {  try {  var data = response.data.json[0]  $('#is\_Swt\_toolname').val(data.swt\_toolname)  $('#is\_Swt\_demo\_site').val(data.swt\_demo\_site)  $('#is\_Giturl').val(data.swt\_github\_url)  $('#is\_Comments').val(data.swt\_comments)  $('#is\_Swt\_function').val(data.swt\_function)  } catch (error) {  alert('작업 중 에러가 발생했습니다.')  }  })  .catch( error => {alert('작업 중 에러가 발생했습니다.');return false;} );  }  .. 코드 생략..  <a href="javascript:" className="bt\_ty bt\_ty2 submit\_ty1 saveclass"  onClick={(e) => this.submitClick('save', e)}>저장</a>  <a href="javascript:" className="bt\_ty bt\_ty2 submit\_ty1 modifyclass"  onClick={(e) => this.submitClick('modify', e)}>수정</a>  .. 코드 생략.. |

6: App.js의 <Route path='/SoftwareView/:swtcode' component={SoftwareView} /> 코드에서 SoftwareView 컴포넌트를 라우팅할 때 swtcode 변수를 함께 전달했다. SoftwareView 컴포넌트에서 swtcode 변수를 사용하려면 props.match.params 객체로 접근해야 한다. 이때 App.js에서 설정한 변수명과 props.match.params 객체 뒤에 붙는 변수명이 일치해야 한다. 전달받은 파라미터를 state 변수 before\_swtcode에 할당한다.

13: 등록 페이지인 경우, line 41~42의 [수정] 버튼을 표시되지 않도록 한다

15~16: 수정 페이지인 경우 node api를 호출하는 함수 callSwToolInfoApi를 실행하고 line 39~40의 [수정] 버튼을 표시되지 않도록 한다

21~23: 리스트 페이지에서 호출했던 node api 경로(/api/Swtool?type=list)을 동일하게 호출한다. 1개의 데이터만 가져와야 하기 때문에 select 쿼리 where절에서 비교할 데이터를 전달해야 한다. line 6에서 할당한 state 변수 before\_swtcode를 is\_Swtcode 변수에 할당에 전달한다.

27~31: api 호출 결과 반환된 response에서 필요한 데이터를 변수명으로 접근해 가져온다. 등록 페이지에서 사용하던 <input> 태그들에 id 값으로 접근해 response 데이터들을 value로 할당한다.

132. NODE 상세 조회 select 쿼리 where절 추가

**• 학습 내용:** 1개의 데이터만 조회할 수 있도록 쿼리에 where절을 추가한다.

**• 힌트 내용:** 리스트 페이지 조회 쿼리를 재사용할 수 있다.

mybatis에서는 특정 파라미터의 값에 따라 쿼리의 일부를 추가하고 제거할 수 있다. 리스트 페이지에서 수정 페이지로 전달한 react\_swtool 테이블의 swt\_code를 where절에 사용한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 | .. 코드 생략..  <select id="selectSwToolsList">  SELECT  swt\_code  , swt\_toolname  , swt\_function  , swt\_imagepath  , swt\_big\_imgpath  , swt\_comments  , swt\_demo\_site  , swt\_manual\_path  , swt\_github\_url  , reg\_date  FROM react.react\_swtool  <if test="is\_Swtcode != null && is\_Swtcode != ''">  WHERE swt\_code = #{is\_Swtcode}  </if>  ORDER BY update\_date DESC  </select>  .. 코드 생략.. |

16~18: 파라미터로 전달된 is\_Swtcode 변수가 null이나 공백이 아니라면 where절을 사용한다. is\_Swtcode 변수를 react\_swtool 테이블의 기본 키인 swt\_code 컬럼 값을 비교하기 때문에 쿼리 실행 결과 1개의 데이터만 조회된다.

리스트 페이지에서 [수정] 버튼을 눌러 수정 페이지로 진입하면, 다음과 같이 조회된 1개의 데이터가 표시되는 것을 확인할 수 있다. 주소 창의 url을 보면 <http://localhost:3000/SoftwareView/> 뒤에 swt\_code 값(USW20200101000000)이 전달된 것을 알 수 있다.

|  |
| --- |
|  |

**<그림 198>**

133. NODE 수정 Api 만들기 1 - update 라우터 분기

**• 학습 내용:** 데이터를 수정하기 위한 post 방식의 api를 구현한다.

**• 힌트 내용:** node 라우터 코드에서 사용할 mapper 정보를 분기 처리한다.

조회, 등록 api를 구현할 때 데이터 연동을 위한 기본 틀이 갖춰졌다. 전달하는 데이터만 달라질 뿐 수정, 삭제 기능도 동일한 틀 안에서 분기 처리해 사용할 수 있다. react에서 호출할 등록 api는 /api/Swtool?type=modify이다. ? 뒤에 파라미터 type 변수의 값만 다를 뿐 호출 url은 동일하다. 그렇기 때문에 server.js 파일은 수정사항이 없다.

routes 폴더의 SwtoolRout.js 파일을 확인한다.

<소스> SwtoolRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 | .. 코드 생략..  }else if(type == 'modify'){  //Swtool 수정  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'SwToolsMapper';//mybatis xml 파일명  req.body.crud = 'update';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'updateSwToolsInfo';    router.use('/', dbconnect\_Module);  next('route')  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: 파라미터로 전달받은 type 값이 modify인 경우, update 쿼리를 실행할 mapper 정보를 body에 할당한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35 | .. 코드 생략..  router.post("/", (req, res) => {  const mybatisMapper = require("mybatis-mapper");  var param = req.body;  //mybatis mapper 경로 설정  mybatisMapper.createMapper(['./models/'+param.mapper+'.xml']);  var time = new Date();  console.log('## '+time+ ' ##');  console.log("\n Called Mapper Name = "+param.mapper);  var format = { language: 'sql', indent: ' ' };  //mysql 쿼리 정보 세팅  var query = mybatisMapper.getStatement(param.mapper, param.mapper\_id, param, format);  console.log("\n========= Node Mybatis Query Log Start =========");  console.log("\* mapper namespce : "+param.mapper+"."+param.mapper\_id+" \*\n");  console.log(query+"\n");  pool.getConnection(function(err,connection){  connection.query(query, function (error, results) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  .. 코드 생략..  connection.release();  console.log("========= Node Mybatis Query Log End =========\n");  });  })  });  module.exports = router; |

dbconnect\_Module.js 파일은 mysql 서버(rds) 접속 정보, mybatis 연동, connection pool에 대한 코드가 구현돼 있다. dbconnect\_Module.js은 모듈화해 공통 코드로 사용하고 SwtoolRout.js 라우터 파일에서 전달받은 파라미터에 따라 mapper 정보를 불러와 쿼리를 실행하고 react 페이지로 response를 전달한다.

8: SwtoolRout.js에서 전달받은 req.body.mapper 값인 SwToolsMapper로 사용할 mapper 경로를 설정한다.

15: SwtoolRout.js에서 전달받은 req.body.mapper\_id 값인 updateSwToolsInfo로 query 변수를 생성한다.

134. NODE 수정 Api 만들기 2 - update 쿼리 추가, response 처리

**• 학습 내용:** mapper 파일에 update 쿼리를 추가하고 실행 결과에 따라 response를 보낸다.

**• 힌트 내용:** update는 select와 달리 react 페이지로 전송할 데이터가 없다.

xml 파일(mapper)에 update 쿼리를 추가하는 방식은 insert 대신 <update> 태그를 사용하는 것 외에는 insert에서 사용한 것과 동일하다. 쿼리를 실행한 후 insert와 마찬가지로 성공 여부를 response로 전달한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24 | .. 코드 생략..  <update id="updateSwToolsInfo">  UPDATE react.react\_swtool  SET  swt\_toolname = #{is\_Swt\_toolname}  , swt\_function = #{is\_Swt\_function}  , swt\_comments = #{is\_Comments}  , swt\_demo\_site = #{is\_Swt\_demo\_site}  , swt\_github\_url = #{is\_Giturl}  <if test="is\_LabelImg != null && is\_LabelImg != ''">  , swt\_imagepath = #{is\_LabelImg}  </if>  <if test="is\_MainImg != null && is\_MainImg != ''">  , swt\_big\_imgpath = #{is\_MainImg}  </if>  <if test="is\_MenualName != null && is\_MenualName != ''">  , swt\_manual\_path = #{is\_MenualName}  </if>  , update\_date = DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , update\_user = #{is\_Email}  WHERE swt\_code = #{is\_beforeSwtcode}  </update>  .. 코드 생략.. |

20: <update> 태그를 사용해 mapperid를 정의한다.

3~22: 실행할 update 쿼리를 작성한다.

11~19: mybatis에서는 전달받은 파라미터로 if문을 사용할 수 있다. 필수 값이 아닌 컬럼 값이 null이나 공백으로 넘어올 경우, 테이블에 삽입하지 않는다. 파라미터 is\_LabelImg, is\_MainImg, is\_MenualName은 공백으로 넘어오는데, 이는 파일과 이미지 업로드 기능을 구현할 때 사용한다.

22: 테이블에서 유일한 값을 갖는 기본 키인 swt\_code의 값과 is\_beforeSwtcode 파라미터가 일치하는 데이터를 찾아 수정한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | .. 코드 생략..  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  if(results != undefined){  string = JSON.stringify(results);  var json = JSON.parse(string);  if (req.body.crud == "select") {  res.send({ json });  }else{  res.send("succ");  }  }else{  res.send("error");  }  .. 코드 생략.. |

12: insert와 동일하게 update 쿼리 실행이 성공하면, response에 succ를 담아 전달한다.

15: update 쿼리를 실행하는 도중 에러가 발생하면 results 변수가 undefined가 된다. 이때 response에 에러를 담아 전달한다.

135. REACT 수정 페이지 만들기

**• 학습 내용:** react에서 node api를 호출해 전달한 데이터로 table의 데이터를 수정한다.

**• 힌트 내용:** api 호출 결과에 따라 사용자에게 적절한 메시지를 표시한다.

react 컴포넌트에서 post 방식으로 axios를 사용해 node api를 호출한다. Insert와 동일하게 update 쿼리에 사용할 데이터를 json 형태로 변경해 전달하고 쿼리가 정상적으로 실행됐다면 완료 후 리스트 페이지로 이동한다. 상세 조회 페이지에서 조회된 데이터가 화면(<input> 태그)에 표시되는데, 이 값들을 수정해 입력하고 update api를 호출하면 데이터가 변경된다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31 32  33  34  35  36  37 | .. 코드 생략..  try {  const response = await fetch('/api/Swtool?type='+type, {  method: 'POST',  headers: {  'Content-Type': 'application/json',  },  body: Json\_form,  });  const body = await response.text();  if(body == "succ"){  if(type == 'save'){  this.sweetalertSucc('Software Tools 등록이 완료됐습니다.', false)  }else if(type == "modify"){  this.sweetalertSucc('Software Tools 수정이 완료됐습니다.', false)  }  setTimeout(function() {  this.props.history.push('/SoftwareList');  }.bind(this),1500  );  }else{  alert('작업 중 에러가 발생했습니다.')  }  } catch (error) {  alert('작업 중 에러가 발생했습니다.')  }  .. 코드 생략..  <form name="frm" id="frm" action="" onsubmit="" method="post" >  <input id="is\_Swtcode" type="hidden" name="is\_Swtcode" />  <input id="is\_Email" type="hidden" name="is\_Email" value="guest" />  <input id="is\_beforeSwtcode" type="hidden" name="is\_beforeSwtcode"  value={this.state.before\_swtcode} />  .. 코드 생략..  <a href="javascript:" className="bt\_ty bt\_ty2 submit\_ty1 modifyclass"  onClick={(e) => this.submitClick('modify', e)}>수정</a>  .. 코드 생략.. |

35~36: [수정] 버튼을 누르면 submitClick 함수를 호출하고 modify 문자열을 파라미터로 보낸다. submitClick 함수가 실행됐을 때 <input> 태그의 값을 가져와 유효성 체크(fnValidate 함수를 실행)하는 것과 line 9의 Json\_form를 만드는 과정은 insert 코드와 동일하다.

3: insert와 동일한 /api/Swtool 경로를 호출하지만, 파라미터 type의 값이 modify로 할당된다.

15~17: update 쿼리가 정상적으로 실행되면, 사용자에게 수정 완료 메시지를 표시한다.

리스트 페이지에서 [수정] 버튼을 누르면, 상세 조회 페이지로 이동한다. 입력 값을 변경한 후 [수정] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 199>**

update 쿼리가 정상적으로 실행되면, 다음과 같이 리스트 페이지로 이동된다. 수정된 데이터가 리스트에 표시되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 200>**

136. NODE 삭제 Api 만들기 1 - delete 라우터 분기

**• 학습 내용:** 데이터를 삭제하기 위한 post 방식의 api를 구현한다.

**• 힌트 내용:** node 라우터 코드에서 사용할 mapper 정보를 분기 처리한다.

조회, 등록 수정 api를 구현할 때 데이터 연동을 위한 기본 틀이 갖춰졌다. 전달하는 데이터만 달라질 뿐 삭제 기능도 동일한 틀 안에서 분기 처리해 사용할 수 있다. react에서 호출할 등록 api는 /api/Swtool?type=delete이다. ? 뒤에 파라미터 type 변수의 값만 다를 뿐, 호출 url은 동일하다. 그렇기 때문에 server.js 파일은 수정사항이 없다.

routes 폴더의 SwtoolRout.js 파일을 확인한다.

<소스> SwtoolRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 | .. 코드 생략..  }else if(type == 'delete'){  //Swtool 삭제  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'SwToolsMapper';//mybatis xml 파일명  req.body.crud = 'delete';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'deleteSwToolsInfo';    router.use('/', dbconnect\_Module);  next('route')  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: 파라미터로 전달받은 type 값이 modify인 경우, update 쿼리를 실행할 mapper 정보를 body에 할당한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35 | .. 코드 생략..  router.post("/", (req, res) => {  const mybatisMapper = require("mybatis-mapper");  var param = req.body;  //mybatis mapper 경로 설정  mybatisMapper.createMapper(['./models/'+param.mapper+'.xml']);  var time = new Date();  console.log('## '+time+ ' ##');  console.log("\n Called Mapper Name = "+param.mapper);  var format = { language: 'sql', indent: ' ' };  //mysql 쿼리 정보 세팅  var query = mybatisMapper.getStatement(param.mapper, param.mapper\_id, param, format);  console.log("\n========= Node Mybatis Query Log Start =========");  console.log("\* mapper namespce : "+param.mapper+"."+param.mapper\_id+" \*\n");  console.log(query+"\n");  pool.getConnection(function(err,connection){  connection.query(query, function (error, results) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  .. 코드 생략..  connection.release();  console.log("========= Node Mybatis Query Log End =========\n");  });  })  });  module.exports = router; |

dbconnect\_Module.js 파일은 mysql 서버(rds) 접속 정보, mybatis 연동, connection pool에 대한 코드가 구현돼 있다. dbconnect\_Module.js은 모듈화해 공통 코드로 사용하고 SwtoolRout.js 라우터 파일에서 전달받은 파라미터에 따라 mapper 정보를 불러와 쿼리를 실행하고 react 페이지로 response를 전달한다.

8: SwtoolRout.js에서 전달받은 req.body.mapper 값인 SwToolsMapper로 사용할 mapper 경로를 설정한다.

15: SwtoolRout.js에서 전달받은 req.body.mapper\_id 값인 deleteSwToolsInfo로 query 변수를 생성한다.

137. NODE 삭제 Api 만들기 2 - delete 쿼리 추가, response 처리

**• 학습 내용:** mapper 파일에 delete 쿼리를 추가하고 실행 결과에 따라 response를 보낸다.

**• 힌트 내용:** select와 달리, delete는 react 페이지로 전송할 데이터가 없다.

xml 파일(mapper)에 delete 쿼리를 추가하는 방식은 <insert>나 <update> 대신 <delete> 태그를 사용하는 것 외에는 동일하다. 쿼리를 실행한 후 insert, update와 마찬가지로 성공 여부를 response로 전달한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6 | .. 코드 생략..  <delete id="deleteSwToolsInfo">  DELETE FROM react.react\_swtool  WHERE swt\_code = #{is\_SwtCd}  </delete>  .. 코드 생략.. |

2: <delete> 태그를 사용해 mapperid를 정의한다.

2~5: 실행할 delete 쿼리를 작성한다. 테이블에서 유일한 값을 갖는 기본 키인 swt\_code의 값과 is\_SwtCd 파라미터가 일치하는 데이터를 찾아 삭제한다.

|  |
| --- |
| NOTE |
| 예제에서는 delete문을 사용해 보기위해 데이터를 삭제했다. 하지만 보통 실무에서는 데이터가 삭제되는 것을 꺼리기 때문에 삭제 플래그 컬럼을 사용한다. 삭제할 때 플래그를 N으로 바꾼다면, 조회할 때는 플래그가 Y인 데이터만 조회하면 된다. |

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | .. 코드 생략..  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  if(results != undefined){  string = JSON.stringify(results);  var json = JSON.parse(string);  if (req.body.crud == "select") {  res.send({ json });  }else{  res.send("succ");  }  }else{  res.send("error");  }  .. 코드 생략.. |

12: insert, update와 동일하게 delete 쿼리 실행이 성공하면, response에 succ를 담아 전달한다.

15: delete 쿼리 실행 중 에러가 발생하면 results 변수가 undefined가 된다. 이때 response에 에러를 담아 전달한다.

138. REACT 리스트 페이지 삭제 기능 구현

**• 학습 내용:** react에서 node api를 호출해 전달한 변숫값으로 table의 데이터를 삭제한다.

**• 힌트 내용:** api 호출 결과에 따라 사용자에게 적절한 메시지를 표시한다.

react 컴포넌트에서 post 방식으로 axios를 사용해 node api를 호출한다. delete 쿼리의 조건절에 사용할 변수를 json 형태로 전달하고 쿼리가 정상적으로 실행됐다면, 리스트 페이지 데이터를 다시 불러온다.

SoftwareToolsManage 폴더의 SoftwareList.js 파일을 확인한다.

<소스> SoftwareList.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31 32  33  34  35  36  36  37  38  39  40 41  42  43  44  45  46  47  48  50  51 | .. 코드 생략..  result.push(  <tr class="hidden\_type">  <td>{data.swt\_toolname}</td>  <td>{data.swt\_function}</td>  <td>{reg\_date}</td>  <td>  <Link to={'/SoftwareView/'+data.swt\_code}  className="bt\_c1 bt\_c2 w50\_b">수정</Link>  <a href="#n" class="bt\_c1 w50\_b" id={data.swt\_code}  onClick={(e) => this.deleteSwtool(e)}>삭제</a>  </td>  </tr>  )  .. 코드 생략..  deleteSwtool = (e) => {  var event\_target = e.target  this.sweetalertDelete('정말 삭제하시겠습니까?', function() {  axios.post('/api/Swtool?type=delete', {  is\_SwtCd : event\_target.getAttribute('id')  })  .then( response => {  this.callSwToolListApi()  }).catch( error => {alert('작업 중 에러가 발생했습니다.');return false;} );  }.bind(this))  }  sweetalertDelete = (title, callbackFunc) => {  Swal.fire({  title: title,  text: "",  icon: 'warning',  showCancelButton: true,  confirmButtonColor: '#3085d6',  cancelButtonColor: '#d33',  confirmButtonText: 'Yes'  }).then((result) => {  if (result.value) {  Swal.fire(  'Deleted!',  '삭제됐습니다.',  'success'  )  }else{  return false;  }  callbackFunc()  })  }  .. 코드 생략.. |

11~12: 리스트에서 [삭제] 버튼을 누르면, deleteSwtool 함수를 실행한다.

19: 실수로 [삭제] 버튼을 눌렀을 때 데이터가 삭제되는 것을 방지하기 위해 한 번 더 사용자에게 확인 메시지를 표시하는 sweetalertDelete 함수를 실행한다.

29~37: sweetalertDelete 함수는 sweetalert2 패키지에서 지원하는 형식을 사용해 알림 창을 표시한다. Swal.fire 함수로 ‘정말 삭제하시겠습니까?’ 메시지를 표시하고 사용자가 [Yes] 버튼을 누르면 then 함수를 실행한다.

38~46: ‘정말 삭제하시겠습니까?’ 알림 창에서 [Yes]를 선택하면, result.value 값이 true가 된다. 이때 ‘삭제됐습니다’ 알림이 표시된다. [Cancel]을 선택하면 result.value 값이 false가 돼 알림 창이 닫힌다.

47: line 19에서 두 번째 파라미터로 전달한 node api 호출 함수를 콜백 함수로 실행한다.

20~22: 데이터 삭제를 위해 node api 경로(/api/Swtool?type=delete)를 호출한다. delete 쿼리 조건절에서 사용할 swt\_code을 line 11 <a> 태그의 id 값에 접근에 가져온다. key가 is\_SwtCd인 json 데이터에 swt\_code 값을 할당하고 api 호출 시 함께 전달한다.

23~25: 삭제 api가 정상적으로 실행됐다면, 조회 api를 호출해 리스트를 다시 표시하는 callSwToolListApi 함수를 실행한다.

리스트 페이지에서 [삭제] 버튼을 누르면, 바로 삭제 api를 호출하지 않는다. 다음과 같이 알림 창을 표시해 사용자에게 한 번 더 확인을 받는다.

|  |
| --- |
|  |

**<그림 201>**

[Yes] 버튼을 누르면 삭제 api를 호출한 후 리스트 조회 api를 호출한다. 이때 리스트에서 데이터가 삭제돼 표시되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 202>**

139. 파일, 이미지 업로드 Api 만들기 - multer 패키지 사용

**• 학습 내용:** multer 패키지를 사용해 파일 업로드 Api를 구현한다.

**• 힌트 내용:** node 서버 경로에 파일을 업로드할 폴더를 생성한다.

문서나 아미지와 같은 파일을 서버 경로에 업로드하기 위해 multer라는 패키지를 사용한다. 등록 수정 node api를 호출할 때는 json 형태로 문자열 데이터를 전달했다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save multer]를 입력하면 다음과 같이 npm이 multer를 설치한다.

|  |
| --- |
|  |

**<그림 203>**

같은 파일을 여러 번 업로드하는 경우, 파일명이 구분되지 않기 때문에 현재 시간 정보를 파일명에 추가해 저장한다. moment 패키지를 사용해 node 서버의 날짜, 시간 정보를 가져올 수 있다. node 경로(C:\react200)에 [npm install --save moment]를 입력하면, 다음과 같이 npm이 moment를 설치한다.

|  |
| --- |
|  |

**<그림 204>**

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 | .. 코드 생략..  var fileuploadRouter = require("./routes/UploadRout");  .. 코드 생략..  app.use("/api/upload", fileuploadRouter);  app.use(express.static("./uploads"));  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

4: /api/upload 경로로 api가 호출되면, line 2에서 지정한 파일 경로로 라우팅한다. UploadRout 라우터에서 업로드할 파일을 전송받아 요청된 node 서버 경로에 업로드한다.

5: express.static 함수를 사용하면 정적 파일 경로(/uploads)를 설정할 수 있다. 정적 파일 경로를 사용하면 파일 사용이 필요할 때 지정한 경로에서만 파일을 찾아 보안상의 이점이 있다. 또 지정 경로(/uploads)를 제외한 짧은 url로 파일 경로를 호출할 수 있다. 예를 들어, C:\react200\uploads\image 경로에 있는 react.jpg라는 이미지를 사용하기 위해 uploads 경로를 제외한 http://localhost:5000/image/react.jpg 경로를 호출하면 된다.

|  |
| --- |
| NOTE |
| 정적 파일이란, image, css, js 파일과 같이 내용이 고정돼 있어서 요청이 왔을 때 별도의 처리 없이 그대로 제공하면 되는 파일이다. |

routes 폴더의 UploadRout.js 파일을 확인한다.

<소스> UploadRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 | var express = require('express');  var router = express.Router();  var upload = require('./fileupload');  var multer = require('multer');  router.post("/", (req, res, next) => {  upload(req, res, function(err) {  if (err instanceof multer.MulterError) {  return next(err);  } else if (err) {  return next(err);  }  console.log('원본 파일명 : ' + req.file.originalname)  console.log('저장 파일명 : ' + req.file.filename)  console.log('크기 : ' + req.file.size)  return res.json({filename:req.file.filename});  });  });  module.exports = router; |

8: post 방식으로 라우터가 호출되면, line 3에서 require한 fileupload 모듈을 실행한다. 모듈에서 작성한 작업이 완료되면 콜백 함수를 실행해 에러 처리와 로그를 출력한다

9~13: 에러가 발생할 경우 콜백 함수를 통해 err 변수로 에러 내용이 반환된다. 에러의 종류가 multer 관련 에러(MulterError)이거나 아닐 경우를 나눠 에러 처리를 할 수 있다.

14~16: fileupload 모듈이 실행되면, request에 파일 정보가 담긴다. 전달된 파일의 정보를 로그로 출력한다. file.originalname은 사용자가 업로드한 파일명이고 file.filename은 fileupload 모듈에서 생성한 업로드 파일명이다.

17: 실제로 업로드된 파일명을 response에 담아 전달한다.

140. REACT 등록 페이지 - 파일, 이미지 업로드 Api 호출하기

**• 학습 내용:** react 페이지에서 파일 업로드 Api 호출 방법을 확인한다.

**• 힌트 내용:** FormData에 파일을 담아 api를 호출하고 파라미터로 파일 저장 경로를 전달한다.

react 페이지에서 선택한 파일을 node api를 통해 node 경로에 업로드한다. 파일이 정상적으로 업로드 됐다면, 파일명과 미리보기 이미지를 표시한다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49  50  51  52  53 54  55  56  57  58  59  60  61  62 63  64  65  66  67  68  69  70  71  72  73  74  75 76  77  78  79  80  81 82  83  84  85  86  87 88  89  90  91  92  93 94  95  96  97  98  99 100  101  102  103  104  105  106  107  108 109  110  111  112  113  114 | .. 코드 생략..  class SoftwareView extends Component {  constructor(props) {  super(props);  this.state = {  before\_swtcode: props.match.params.swtcode,  selectedFile: null,  }  }  .. 코드 생략..  handleFileInput(type, e){  if(type =='file'){  $('#imagefile').val(e.target.files[0].name)  }else if(type =='file2'){  $('#imagefile2').val(e.target.files[0].name)  }else if(type =='manual'){  $('#manualfile').val(e.target.files[0].name)  }  this.setState({  selectedFile : e.target.files[0],  })  setTimeout(function() {  if(type =='manual'){  this.handlePostMenual()  }else{  this.handlePostImage(type)  }  }.bind(this),1  );  }  handlePostMenual(){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/swmanual/", formData).then(res => {  this.setState({menualName : res.data.filename})  $('#is\_MenualName').remove()  $('#upload\_menual').prepend('<input id="is\_MenualName" type="hidden"'  +'name="is\_MenualName" value="/swmanual/'+this.state.menualName+'"}/>')  }).catch(error => {  alert('작업 중 에러가 발생했습니다.', error, 'error', '닫기')  })  }  handlePostImage(type){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/image/", formData).then(res => {  if(type =='file'){  this.setState({fileName : res.data.filename})  $('#is\_MainImg').remove()  $('#uploadimg').remove()  $('#upload\_img').prepend('<img id="uploadimg" src="/image/'  +this.state.fileName+'"/>')  $('#upload\_img').prepend('<input id="is\_MainImg" type="hidden"'  +'name="is\_MainImg" value="/image/'+this.state.fileName+'"}/>')  }else if(type =='file2'){  this.setState({fileName2 : res.data.filename})  $('#is\_LabelImg').remove()  $('#uploadimg2').remove()  $('#upload\_img2').prepend('<img id="uploadimg2" src="/image/'  +this.state.fileName2+'"/>')  $('#upload\_img2').prepend('<input id="is\_LabelImg" type="hidden"'  +'name="is\_LabelImg" value="/image/'+this.state.fileName2+'"}/>')  }  }).catch(error => {  alert('작업 중 에러가 발생했습니다.')  })  }  .. 코드 생략..  <tr class="div\_tb\_tr fileb">  <th>  매뉴얼 파일 #1  </th>  <td class="fileBox fileBox\_w1">  <label for="uploadBtn1" class="btn\_file"> 파일선택</label>  <input type="text" id="manualfile" class="fileName fileName1"  readonly="readonly" placeholder="선택된 파일 없음"/>  <input type="file" id="uploadBtn1" class="uploadBtn uploadBtn1"  onChange={e => this.handleFileInput('manual',e)}/>  <div id="upload\_menual">  </div>  </td>  </tr>  <tr>  <th>  메인 이미지  </th>  <td className="fileBox fileBox1">  <label htmlFor='imageSelect' className="btn\_file"> 파일선택</label>  <input type="text" id="imagefile" className="fileName fileName1"  readOnly="readonly" placeholder="선택된 파일 없음"/>  <input type="file" id="imageSelect" className="uploadBtn uploadBtn1"  onChange={e => this.handleFileInput('file',e)}/>  <div id="upload\_img">  </div>  </td>  </tr>  <tr>  <th>  라벨 이미지  </th>  <td className="fileBox fileBox2">  <label htmlFor='imageSelect2' className="btn\_file"> 파일선택</label>  <input type="text" id="imagefile2" className="fileName fileName1"  readOnly="readonly" placeholder="선택된 파일 없음"/>  <input type="file" id="imageSelect2" className="uploadBtn uploadBtn1"  onChange={e => this.handleFileInput('file2',e)}/>  <div id="upload\_img2">  </div>  </td>  </tr>  .. 코드 생략.. |

80, 94, 108: [파일 선택] 버튼을 누르고 업로드할 파일 또는 이미지를 선택하면 handleFileInput 함수를 실행한다. 누르는 [파일 선택] 버튼(매뉴얼 파일, 메인 이미지 라벨 이미지)에 따라 서로 다른 파라미터(file, file2, manual)를 전달한다.

|  |
| --- |
| NOTE |
| type이 file인 <input> 태그는 클릭하면, 파일 선택 창이 표시된다. 이때 업로드할 파일을 선택하고 [열기] 버튼을 누르면 웹 브라우저에 파일이 저장되고 그 파일 정보는 <input> 태그에 남는다. handleFileInput 함수를 실행할 때 파라미터로 태그에 대한 정보(e)가 전달된다. line 21의 e.target.files[0]로 웹 브라우저에 업로드된 파일에 접근할 수 있다. |

13~19: e.target.files[0].name으로 업로드한 파일의 이름을 가져온다. 파라미터(file, file2, manual)에 따라 [파일 선택] 버튼 옆에 표시되는 <input> 태그에 파일명을 할당해 표시한다.

23~30: 매뉴얼 파일 업로드인 경우와 이미지 파일 업로드인 경우를 나눠 실행 함수를 분기한다. 함수(handlePostMenual, handlePostImage) 실행 시 state 변수 selectedFile 값을 참조한다. line 21에서 실제 파일이 selectedFile에 할당하기 전에 함수(handlePostMenual, handlePostImage)가 실행되면 selectedFile 값을 참조할 수 없기 때문에 setTimeout 함수로 파일이 할당되기까지 지연 시간을 설정한다.

34~35: formData 객체를 생성하고 append 함수를 사용해 key 값으로 file을 value 값으로 업로드할 파일(this.state.selectedFile)을 할당한다.

|  |
| --- |
| NOTE |
| formData는 자바스크립트에서 서버에 데이터를 전달해주는 API다. formData 객체를 생성하고 append 함수로 key, value를 할당하면, 문자열과 파일을 전송할 수 있다. |

36~37: axios를 사용해 post 방식으로 node api(/api/upload?type=uploads/swmanual/)를 호출한다. 파라미터로 업로드 파일을 할당한 FormData를 넘긴다. 파일이 정상적으로 업로드되면, 반환된 업로드 파일명(res.data.filename)을 state 변수 menualName에 할당한다.

38: 여러 번 매뉴얼 파일을 추가하는 경우, 기존에 있던 영역을 삭제한다.

40~41: id가 upload\_menual인 <div> 태그 안에 hidden type의 <input> 태그를 prepend 함수로 추가한다. value 값으로는 파일이 업로드되는 경로가 할당된다. 이 <input> 태그는 <form> 태그 안에 추가되는데, 저장이나 수정 api를 호출할 때 name과 value 값이 함께 전달된다.

49: 매뉴얼 파일 업로드와 같은 api를 호출하지만, 업로드 경로를 할당한 파라미터 type의 값이 uploads/image/이다.

50~66: type 값(메인 이미지: file, 라벨 이미지: file2)에 따라 분기 처리한다.

51: 이미지가 정상적으로 업로드되면, filename이 response로 반환된다. res.data.filename을 state 변수 fileName에 할당한다.

54~55: 업로드된 이미지의 미리보기 영역(<img> 태그)을 line 96 id가 upload\_img인 <div> 태그 안에 추가한다.

56~57: 이미지 업로드 경로가 value로 할당된 <input> 태그는 <form> 태그 안에 추가된다. 저장이나 수정 api를 호출할 때 name과 value 값이 함께 전달된다.

52~53: 여러 번 이미지를 추가하는 경우, 기존에 있던 영역을 삭제한다.

59~65: 라벨 이미지 분기 영역으로, 메인 이미지와 동일한 구조로 동작한다.

매뉴얼 파일, 메인 이미지 라벨 이미지를 추가하면 다음과 같이 input 영역에 업로드 파일명이 표시된다. 메인 이미지 라벨 이미지의 경우 미리보기 이미지가 표시된다.

|  |
| --- |
|  |

**<그림 206>**

업로드 경로로 지정한 폴더를 확인해보면, 다음과 같이 실제 업로드한 파일과 이미지가 추가된 것을 확인할 수 있다.

|  |
| --- |
| **<그림 207>**    **<그림 208>** |

141. REACT 등록 페이지 - 업로드 경로를 db에 insert하기

**• 학습 내용:** node 경로에 업로드한 파일의 경로를 db에 insert한다.

**• 힌트 내용:** [저장] 버튼을 눌렀을 때 업로드 파일의 경로가 insert된다.

실제 파일을 서버 경로에 업로드하고 db에 업로드 경로를 삽입해야 한다. 조회, 수정 페이지에서 db에 저장된 경로를 조회하면 이미지를 확인할 수 있다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38 39  40  41  42  43  44 45  46  47  48  49  50  51  52  53 54  55  56  57  58  59  60  61  62 63  64  65  66 67 | .. 코드 생략..  handlePostMenual(){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/swmanual/", formData).then(res => {  .. 코드 생략..  $('#upload\_menual').prepend('<input id="is\_MenualName" type="hidden"'  +'name="is\_MenualName" value="/swmanual/'+this.state.menualName+'"}/>')  }).catch(error => {  alert('작업 중 에러가 발생했습니다.', error, 'error', '닫기')  })  }  handlePostImage(type){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/image/", formData).then(res => {  if(type =='file'){  .. 코드 생략..  $('#upload\_img').prepend('<input id="is\_MainImg" type="hidden"'  +'name="is\_MainImg" value="/image/'+this.state.fileName+'"}/>')  }else if(type =='file2'){  .. 코드 생략..  $('#upload\_img2').prepend('<input id="is\_LabelImg" type="hidden"'  +'name="is\_LabelImg" value="/image/'+this.state.fileName2+'"}/>')  }  }).catch(error => {  alert('작업 중 에러가 발생했습니다.')  })  }  .. 코드 생략..  <form name="frm" id="frm" action="" onsubmit="" method="post" >  .. 코드 생략..  <tr class="div\_tb\_tr fileb">  <th>  매뉴얼 파일 #1  </th>  <td class="fileBox fileBox\_w1">  .. 코드 생략..  <div id="upload\_menual">  </div>  </td>  </tr>  <tr>  <th>  메인 이미지  </th>  <td className="fileBox fileBox1">  .. 코드 생략..  <div id="upload\_img">  </div>  </td>  </tr>  <tr>  <th>  라벨 이미지  </th>  <td className="fileBox fileBox2">  .. 코드 생략..  <div id="upload\_img2">  </div>  </td>  </tr>  .. 코드 생략..  </form>  .. 코드 생략.. |

8~9: 매뉴얼 파일을 추가하면, line 41의 id가 upload\_menual인 <div> 태그에 <input> 태그가 추가된다.

21~22: 메인 이미지를 추가하면, line 51의 id가 upload\_img인 <div> 태그에 <input> 태그가 추가된다.

25~26: 라벨 이미지를 추가하면, line 61의 id가 upload\_img2인 <div> 태그에 <input> 태그가 추가된다.

33~66: 추가된 <input> 태그는 <form> 태그 안에 위치하기 때문에 저장 api를 호출할 때 업로드 경로 정보가 함께 전송된다.

매뉴얼 파일, 메일 이미지 라벨 이미지를 등록하고 [저장] 버튼을 누른다. insert 쿼리를 실행할 때 파일 업로드 경로가 삽입되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 209>**

142. REACT 상세 조회 페이지 - 파일, 이미지명 표시하기

**• 학습 내용:** db에서 파일 경로 데이터를 조회해 상세 조회 페이지에 표시한다.

**• 힌트 내용:** 조회한 파일 경로로 화면에 파일명과 미리보기 이미지를 표시할 수 있다.

리스트 페이지에서 [수정] 버튼을 눌러 상세 조회 페이지로 진입한다. Select 쿼리를 실행해 기존에 업로드했던 파일과 이미지의 경로를 불러온다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37 | .. 코드 생략..  callSwToolInfoApi = async () => {  axios.post('/api/Swtool?type=list', {  is\_Swtcode: this.state.before\_swtcode,  })  .then( response => {  try {  var data = response.data.json[0]  $('#is\_Swt\_toolname').val(data.swt\_toolname)  $('#is\_Swt\_demo\_site').val(data.swt\_demo\_site)  $('#is\_Giturl').val(data.swt\_github\_url)  $('#is\_Comments').val(data.swt\_comments)  $('#is\_Swt\_function').val(data.swt\_function)  var manualName = data.swt\_manual\_path.replace('/swmanual/','')  var fileName = data.swt\_big\_imgpath.replace('/image/','')  var fileName2 = data.swt\_imagepath.replace('/image/','')  $('#upload\_img').prepend('<img id="uploadimg" src="'+data.swt\_big\_imgpath+'"/>')  $('#upload\_img2').prepend('<img id="uploadimg2" src="'+data.swt\_imagepath+'"/>')  $('#imagefile').val(fileName)  $('#imagefile2').val(fileName2)  $('#manualfile').val(manualName)  if($('#uploadimg').attr('src').indexOf("null") > -1){  $('#uploadimg').hide()  }  if($('#uploadimg2').attr('src').indexOf("null") > -1){  $('#uploadimg2').hide()  }  } catch (error) {  alert('작업 중 에러가 발생했습니다.')  }  })  .catch( error => {alert('작업 중 에러가 발생했습니다.');return false;} );  }  .. 코드 생략.. |

15~17: db에서 조회한 데이터는 파일 경로와 파일명이 모두 저장돼 있다. 파일명을 따로 표시하기 위해서 파일 경로를 공백으로 대체해 변수에 할당한다.

18~19: 메인 이미지와 라벨 이미지의 미리보기 이미지를 표시하기 위해 <img> 태그 src 속성에 조회한 파일 경로를 할당한다. 생성한 <img> 태그를 id가 upload\_img, upload\_img2인 <div> 태그 안에 삽입한다.

21~23: 매뉴얼 파일, 메인 이미지 라벨 이미지의 [파일 선택] 버튼 옆에 있는 <input> 태그에 파일명을 value 값으로 할당해 표시한다.

25~30: 이미지가 없는 경우, <img> 태그를 숨긴다.

리스트 페이지에서 [수정] 버튼을 눌러 상세 조회 페이지에 진입하면, 다음과 같이 이전에 등록한 파일의 파일명과 미리보기 이미지가 표시된다.

|  |
| --- |
|  |

**<그림 210>**

실행되는 select 쿼리를 보면, 파일이 위치한 경로를 조회하는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 211>**

143. REACT 수정 페이지 - 파일, 이미지 업로드 Api 호출하기

**• 학습 내용:** 상세 조회 페이지에서 파일, 이미지를 수정한다.

**• 힌트 내용:** 새로운 파일, 이미지를 업로드하고 수정 api를 호출한다.

파일, 이미지 수정 기능은 db를 조회해 초깃값을 화면에 표시하는 것 외에 등록과 동일하다. 수정 페이지에서 이미지를 변경하면, 기존 이미지 미리보기(<img> 태그) 영역을 삭제하고 새로운 미리보기 영역을 추가한다.

SoftwareToolsManage 폴더의 SoftwareView.js 파일을 확인한다.

<소스> SoftwareView.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34  35  36  37  38  39 40  41  42  43 | .. 코드 생략..  callSwToolInfoApi = async () => {  axios.post('/api/Swtool?type=list', {  is\_Swtcode: this.state.before\_swtcode,  })  .then( response => {  try {  var data = response.data.json[0]  .. 코드 생략..  var manualName = data.swt\_manual\_path.replace('/swmanual/','')  var fileName = data.swt\_big\_imgpath.replace('/image/','')  var fileName2 = data.swt\_imagepath.replace('/image/','')  $('#upload\_img').prepend('<img id="uploadimg" src="'+data.swt\_big\_imgpath+'"/>')  $('#upload\_img2').prepend('<img id="uploadimg2" src="'+data.swt\_imagepath+'"/>')  .. 코드 생략..  handlePostMenual(){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/swmanual/", formData).then(res => {  this.setState({menualName : res.data.filename})  $('#is\_MenualName').remove()  $('#upload\_menual').prepend('<input id="is\_MenualName" type="hidden"'  +'name="is\_MenualName" value="/swmanual/'+this.state.menualName+'"}/>')  }).catch(error => {  alert('작업 중 에러가 발생했습니다.', error, 'error', '닫기')  })  }  handlePostImage(type){  const formData = new FormData();  formData.append('file', this.state.selectedFile);  return axios.post("/api/upload?type=uploads/image/", formData).then(res => {  if(type =='file'){  this.setState({fileName : res.data.filename})  $('#is\_MainImg').remove()  $('#uploadimg').remove()  $('#upload\_img').prepend('<img id="uploadimg" src="/image/'  +this.state.fileName+'"/>')  $('#upload\_img').prepend('<input id="is\_MainImg" type="hidden"'  +'name="is\_MainImg" value="/image/'+this.state.fileName+'"}/>')  }else if(type =='file2'){  .. 코드 생략.. |

14~15: line 3에서 상세 페이지에 표시할 데이터를 조회해 미리보기 이미지(<img> 태그) 영역을 추가한다. 이때 조회한 파일 경로 정보(swt\_imagepath, swt\_big\_imgpath, swt\_manual\_path)는 사용하지 않는다. 파일 변경을 하지 않는다면 기존의 db에 저장된 값을 유지하면 되고, 변경한다면 새로운 파일 경로로 업데이트하면 되기 때문이다.

21: 여러 번 매뉴얼 파일을 업로드하는 경우, 가장 최근에 올린 파일 1개의 경로만 db에 저장해야 한다. 매뉴얼 파일 경로가 할당된 <input> 태그가 있다면 삭제한다.

22~23: 새로운 매뉴얼 파일을 추가하면, 매뉴얼 파일 경로가 할당된 <input> 태그를 추가한다. 이 태그는 <form> 태그 안에 삽입돼 파일 경로가 수정 api 호출 시 함께 전송된다.

36~37: 여러 번 메인 이미지를 업로드하는 경우, 기존에 추가된 이미지 미리보기 영역과 이미지 경로가 할당된 <input> 태그를 삭제한다.

38~39: 새로운 업로드한 메인 이미지를 미리보기 영역에 표시한다.

40~41: 새로운 메인 이미지가 업로드되면, 이미지 파일 경로가 할당된 <input> 태그를 <form> 태그 안에 삽입한다. <input> 태그에 value로 할당된 이미지 경로가 수정 api 호출 시 함께 전송된다.

42: 라벨 이미지의 수정 기능도 메인 이미지와 동일하게 동작한다.

수정 페이지에서 새로운 이미지를 추가하면, 추가된 이미지로 미리보기 이미지가 변경된다.

|  |
| --- |
|  |

**<그림 212>**

144. REACT 수정 페이지 - 업로드 경로 db에 업데이트하기

**• 학습 내용:** 수정한 파일, 이미지의 경로로 db 정보를 변경한다.

**• 힌트 내용:** 수정 api를 통해 파일, 이미지 경로를 전달받아 update문에 사용한다.

파일, 이미지를 업로드하고 db에 저장된 업로드 경로를 수정하지 않는다면, 상세 페이지에서 새로 추가한 파일과 이미지가 조회되지 않는다. 수정 api 호출 시 파라미터로 전달한 업로드 경로를 update에 넣어 데이터를 변경한다.

models 폴더의 SwToolsMapper.xml 파일을 확인한다.

<소스> SwToolsMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24 | .. 코드 생략..  <update id="updateSwToolsInfo">  UPDATE react.react\_swtool  SET  swt\_toolname = #{is\_Swt\_toolname}  , swt\_function = #{is\_Swt\_function}  , swt\_comments = #{is\_Comments}  , swt\_demo\_site = #{is\_Swt\_demo\_site}  , swt\_github\_url = #{is\_Giturl}  <if test="is\_LabelImg != null && is\_LabelImg != ''">  , swt\_imagepath = #{is\_LabelImg}  </if>  <if test="is\_MainImg != null && is\_MainImg != ''">  , swt\_big\_imgpath = #{is\_MainImg}  </if>  <if test="is\_MenualName != null && is\_MenualName != ''">  , swt\_manual\_path = #{is\_MenualName}  </if>  , update\_date = DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , update\_user = #{is\_Email}  WHERE swt\_code = #{is\_beforeSwtcode}  </update>  .. 코드 생략.. |

11~19: 라벨 이미지(is\_LabelImg), 메인 이미지(is\_MainImg), 매뉴얼 파일 경로(is\_MenualName) 파라미터를 각각의 컬럼에 할당한다. 이미지나 파일 변경 없이 [수정] 버튼을 눌렀다면, 파라미터가 빈 값으로 전달되기 때문에 해당 컬럼을 업데이트하지 않는다.

[수정] 버튼을 누르면 update 쿼리가 실행돼 업로드 경로가 수정된다. 다음 쿼리는 메인 이미지 경로(swt\_big\_imgpath)만 수정했기 때문에 라벨 이미지 경로(swt\_imagepath)와 매뉴얼 파일 경로(swt\_manual\_path)는 업데이트되지 않았다.

|  |
| --- |
|  |

**<그림 213>**

145. 회원 정보 table 생성하기

**• 학습 내용:** 회원 가입 데이터를 insert할 db 테이블을 생성한다.

**• 힌트 내용:** workbench에서 rds 서버에 접속해 테이블을 생성한다.

회원 정보에는 각각의 회원을 구분할 수 있는 유일한 값이 필요하다. 실제 사용 중인 이메일 주소를 유일한 컬럼으로 생성하고 웹 사이트의 아이디로 사용한다.

workbench에서 다음 쿼리를 실행한다.

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | use react;  CREATE TABLE `react\_user` (  `username` varchar(100) DEFAULT NULL COMMENT ' 사용자 이름',  `userorg` varchar(100) DEFAULT NULL COMMENT '소속기관',  `useremail` varchar(100) COMMENT '이메일',  `userpassword` varchar(100) DEFAULT NULL COMMENT '로그인 비밀번호',  `usermajor` varchar(100) DEFAULT NULL COMMENT '전공',  `userphone` varchar(100) DEFAULT NULL COMMENT '휴대전화번호',  `userflag` varchar(100) DEFAULT NULL COMMENT '승인여부',  `reg\_date` varchar(100) DEFAULT NULL COMMENT '등록날짜',  `reg\_user` varchar(100) DEFAULT NULL COMMENT '등록자',  `update\_date` varchar(100) DEFAULT NULL COMMENT '수정날짜',  `update\_user` varchar(100) DEFAULT NULL COMMENT '수정자',  PRIMARY KEY (`useremail`)  );  ALTER TABLE react.react\_user convert to charset utf8; |

1: 웹 사이트에서 사용하는 스키마인 react를 사용하기 위해 use 명령어를 실행한다.

2: create table 쿼리를 실행해 회원 정보 테이블(react\_user)를 생성한다.

15: 테이블에서 유일한 값을 갖는 usermail 컬럼을 기본 키로 지정한다. usermail 컬럼의 값이 웹 사이트의 아이디가 된다.

17: 테이블에 한글 데이터 삽입을 할 수 있도록 charset을 utf8로 변경한다.

workbench에서 세미콜론(;)을 기준으로 3개의 쿼리를 순서대로, 다음과 같이 실행한다.

|  |
| --- |
|  |

146. NODE 회원 가입 Api 만들기 1 - bcrypt로 암호화하기

**• 학습 내용:** bcrypt 패키지로 회원 비밀번호를 암호화할 수 있다.

**• 힌트 내용:** bcrypt 패키지의 salt 기능을 사용하면, 좀 더 안전하게 암호화할 수 있다.

회원 가입을 하면 다양한 데이터를 입력받게 된다. 이 중 로그인 비밀번호를 그대로 db에 넣어 보관하는 것은 매우 위험하다. 비밀번호는 복호화할 필요가 없기 때문에 해시 함수로 암호화한다. 해시 알고리즘은 암호화만 가능하고 복호화는 불가능한데, bcrypt 패키지를 사용하면 간편하게 해시 암호화를 구현할 수 있다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save bcrypt]를 입력하면 다음과 같이 npm이 bcrypt를 설치한다.

|  |
| --- |
|  |

**<그림 215>**

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | .. 코드 생략..  var usersRouter = require("./routes/UsersRout");  var app = express();  .. 코드 생략..  app.use("/api/register", usersRouter);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

6: /api/register 경로로 api가 호출되면, line 2에서 지정한 회원 정보 라우터 파일 경로로 라우팅한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34 | .. 코드 생략..  const bcrypt = require('bcrypt');  const saltRounds = 10;  .. 코드 생략..  router.post('/', (req, res, next) => {  var type = req.query.type;  if(type == "signup"){  //회원 가입 정보 삽입  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');    //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'insert';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'insertUser';    var myPlaintextPassword = req.body.is\_Password;  if(myPlaintextPassword != '' && myPlaintextPassword != undefined ){  bcrypt.genSalt(saltRounds, function(err, salt) {  bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {  req.body.is\_Password = hash;  router.use('/', dbconnect\_Module);  next('route')  });  });  }else{  router.use('/', dbconnect\_Module);  next('route')  }  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  .. 코드 생략.. |

2~3: bcrypt 패키지를 require해 사용할 수 있도록 한다. bcrypt에서 지원하는 salt의 크기를 ‘10’으로 설정한다.

|  |
| --- |
| NOTE |
| salt는 해커가 비밀번호를 유추하기 어렵도록 한 번 더 암호화하는 문자열이다. bcrypt 패키지 내부적으로 salt의 크기만큼의 임의의 문자열을 생성한다. bcript의 hash 함수와 salt를 사용해 암호화한 결과는 hash 함수만으로 암호화한 결과보다 복잡해 복호화가 어렵다. |

14~16: type 파라미터의 값이 signup일 경우, 회원 정보를 insert하는 mapper 정보를 req.body에 할당한다.

18~19: req.body.is\_Password는 암호화되지 않은 비밀번호다. 값이 있다면 brypt 암호화를 실행한다.

20: genSalt 함수로 line 3에서 지정한 크기의 salt를 생성한다.

21: hash 함수로 salt 값을 사용해 암호화되지 않은 비밀번호(myPlaintextPassword)를 암호화한다. 암호화된 비밀번호가 콜백 함수에서 hash 변수로 반환된다.

22: 암호화된 비밀번호(hash)를 다시 req.body.is\_Password 변수에 할당해 다음 라우터(dbconnect\_Module)로 전달한다.

147. NODE 회원 가입 Api 만들기 2 - insert 쿼리 추가

**• 학습 내용:** 회원 정보 mapper 파일에 insert 쿼리를 추가하고 회원 정보를 각 컬럼에 할당한다.

**• 힌트 내용:** 전달받은 파라미터들을 데이터 형태에 맞게 삽입한다.

회원 가입 페이지에서 나눠 전달한 휴대전화 번호와 이메일 주소를 mysql CONCAT 함수를 사용해 가공한 후 삽입한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  25  26  27  28 29  30  31  32  33 | .. 코드 생략..  <insert id="insertUser">  INSERT INTO react.react\_user  (  username  , userorg  , useremail  , userpassword  , usermajor  , userphone  , userflag  , reg\_date  , reg\_user  , update\_date  , update\_user  )  VALUES (  #{is\_Username}  , #{is\_Organization}  , CONCAT(#{is\_Useremail1}, '@', #{is\_Useremail2})  , #{is\_Password}  , #{is\_Usermajor}  , CONCAT(#{is\_Userphone1}, '-', #{is\_Userphone2},'-', #{is\_Userphone3})  , 'Y'  , DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , CONCAT(#{is\_Useremail1}, '@', #{is\_Useremail2})  , DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  , CONCAT(#{is\_Useremail1}, '@', #{is\_Useremail2})  )  </insert>  .. 코드 생략.. |

12: userflag 컬럼은 데이터를 정의하는 것에 따라 다양하게 활용할 수 있다. 회원 탈퇴, 관리자 계정, 비활성 계정 등에 목적에 맞게 각각의 플래그 값을 정의해 사용할 수 있다.

|  |
| --- |
| NOTE |
| mysql CONCAT 함수를 사용하면 여려 개의 문자열을 연결할 수 있다. |

21: 이메일 아이디, @, 이메일 도메인을 CONCAT 함수로 연결해 삽입한다.

22: bcypt로 암호화된 비밀번호가 할당된다.

25: 3개의 변수로 나눠 전달된 휴대전화 번호 변수를 - 부호로 연결해 삽입한다.

148. REACT 회원 가입 페이지 만들기 1 - 입력 form 만들기

**• 학습 내용:** 회원 가입 정보를 입력받는 태그들을 <form> 태그로 감싼다.

**• 힌트 내용:** 입력 유형에 따라 추가되는 태그와 속성을 확인한다.

회원 정보를 저장하는 node api 호출 시 전달할 변수들을 <form> 태그 안에 위치시킨다. <form> 태그로 감싸여진 <input> 태그와 <select> 태그의 데이터는 별도의 변수 선언 없이 편리하게 파라미터로 전달할 수 있다.

components 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9 10  11  12  13 14 15 16 17 | .. 코드 생략..  import Register from './Register/Register';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/register' component={Register} />  <Footer/>  </div>  );  }  }  .. 코드 생략.. |

11: 회원 가입 페이지는 /register 경로로 호출하고 line 2에 선언된 Register 폴더의 Register.js로 라우팅한다.

Register 폴더의 Register.js 파일을 확인한다.

<소스> Register.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50  51 52 53  54 | .. 코드 생략..  mustNumber = (id) => {  var pattern1 = /[0-9]/;  var str = $('#'+id).val();  if(!pattern1.test(str.substr(str.length - 1, 1))){  $('#'+id).val(str.substr(0, str.length-1));  }  }  .. 코드 생략..  <form method="post" name="frm">  .. 코드 생략..  <tr className="re\_email">  <th>이메일</th>  <td>  <input id="email\_val" type="text" name="is\_Useremail1"  placeholder="이메일을 입력해주세요." onKeyPress={this.emailKeyPress}/>  <span className="e\_goll">@</span>  <select id="email2\_val" name="is\_Useremail2" className="select\_ty1">  <option value="">선택하세요</option>  <option value='naver.com'>naver.com</option>  <option value='hanmail.net'>hanmail.net</option>  .. 코드 생략..  </select>  </td>  </tr>  <tr>  <th>비밀번호</th>  <td>  <input id="pwd\_val" type="password" name="is\_Password"  placeholder="비밀번호를 입력해주세요." onKeyPress={this.pwdKeyPress} />  </td>  </tr>  .. 코드 생략..  <tr className="tr\_tel">  <th>핸드폰</th>  <td>  <select id="phone1\_val" name="is\_Userphone1" className="select\_ty1">  <option value="">선택</option>  <option value="010">010</option>  <option value="011">011</option>  .. 코드 생략..  </select>  <span className="tel\_dot">-</span>  <input id="phone2\_val" name="is\_Userphone2" max="9999"  maxlength="4" onChange={(e) => this.mustNumber("phone2\_val")}/>  <span className="tel\_dot">-</span>  <input id="phone3\_val" name="is\_Userphone3" max="9999"  maxlength="4" onChange={(e) => this.mustNumber("phone3\_val")}/>  </td>  </tr>  .. 코드 생략..  </form>  .. 코드 생략.. |

11~53: node api 호출 시에 <input>, <select> 태그의 입력 값들을 한 번에 전달하기 위해 <form> 태그로 감싸준다.

16~17: 이메일의 @ 앞부분을 <input> 태그에 입력한다. 다음 예제에 나오는 유효성 체크에서 input 값을 입력하지 않고 [회원 가입] 버튼을 누르면 <input> 태그에 빨간색 테두리가 생긴다. emailKeyPress 함수는 입력 창에 키를 입력했을 때 실행되고 빨간색 테두리를 제거한다.

19~22: 이메일의 @뒷부분은 <select>, <option> 태그로 선택한다.

30~31: 비밀번호는 type이 password인 <input> 태그에 입력해야 입력한 값이 표시되지 않는다. pwdKeyPress 함수는 유효성 검사에서 생기는 <input> 태그의 빨간색 테두리를 제거한다.

38~41: 휴대폰번호 앞자리를 <select>, <option> 태그로 선택한다.

44~49: 휴대폰 번호의 중간 네 자리와 마지막 네 자리를 나눠 입력받는다. <input> 태그에 max와 maxlength 속성을 추가하면, 숫자의 입력 범위를 지정할 수 있다. Max 값이 9999이면 <input> 태그에 0000부터 9999까지의 숫자를 입력할 수 있다. maxlength가 4라면 입력 값을 최대 네 자리까지 입력할 수 있다. <input> 태그에 값을 입력할 때마다 line 2의 mustNumber 함수를 호출한다.

5~6: pattern1은 숫자를 의미한다. 입력된 글자가 숫자가 아니라면, 가장 마지막에 추가된 문자를 제외하고 <input> 태그에 남긴다. 예를 들어, 1a를 입력했다면 1만 남게 된다.

|  |
| --- |
|  |

**<그림 216>**

149. REACT 회원 가입 페이지 만들기 2 - 유효성 체크하기

**• 학습 내용:** 입력받은 회원 정보가 유효한 값인지 체크 표시를 한다.

**• 힌트 내용:** 입력 값이 유효하지 않다면, 사용자에게 재입력 메시지를 표시한다.

db에 정합성 높은 데이터를 삽입하기 위해서 node api 호출 전 입력받은 값이 유효한지 검사한다. 하나의 입력 값이라도 잘못된 데이터가 들어왔다면, node api 호출을 하지 않고 재입력 사유에 대해 메시지를 표시한다.

Register 폴더의 Register.js 파일을 확인한다.

<소스> Register.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50  51 52 53  54  55  56  57 58 59  60 | .. 코드 생략..  submitClick = async (type, e) => {  .. 코드 생략..  this.fnValidate = (e) => {  var pattern1 = /[0-9]/;  var pattern2 = /[a-zA-Z]/;  var pattern3 = /[~!@#$%^&\*()\_+|<>?:{}]/;  .. 코드 생략..  if(this.pwd\_val\_checker ==='') {  $('#pwd\_val').addClass('border\_validate\_err');  this.sweetalert('비밀번호를 입력해주세요.', '', 'info', '닫기')  return false;  }  if(this.pwd\_val\_checker !=='') {  var str = this.pwd\_val\_checker;  if(str.search(/\s/) !== -1) {  $('#pwd\_val').addClass('border\_validate\_err');  this.sweetalert('비밀번호 공백을 제거해 주세요.', '', 'info', '닫기')  return false;  }  if(!pattern1.test(str) || !pattern2.test(str) || !pattern3.test(str)  || str.length < 8 || str.length > 16) {  $('#pwd\_val').addClass('border\_validate\_err');  this.sweetalert('8~16자 영문 대 소 문자\n숫자, 특수 문자를 사용하세요.', '', 'info', '닫기')  return false;  }  }  $('#pwd\_val').removeClass('border\_validate\_err');  if(this.pwd\_cnf\_val\_checker ==='') {  $('#pwd\_cnf\_val').addClass('border\_validate\_err');  this.sweetalert('비밀번호 확인을 입력해주세요.', '', 'info', '닫기')  return false;  }  if(this.pwd\_val\_checker !== this.pwd\_cnf\_val\_checker) {  $('#pwd\_val').addClass('border\_validate\_err');  $('#pwd\_cnf\_val').addClass('border\_validate\_err');  this.sweetalert('비밀번호가 일치하지 않습니다.', '', 'info', '닫기')  return false;  }  $('#pwd\_cnf\_val').removeClass('border\_validate\_err');  .. 코드 생략..  if(this.phone1\_val\_checker ==='' || this.phone2\_val\_checker ===''  || this.phone3\_val\_checker ==='') {  $('#phone1\_val').addClass('border\_validate\_err');  $('#phone2\_val').addClass('border\_validate\_err');  $('#phone3\_val').addClass('border\_validate\_err');  this.sweetalert('휴대전화 번호를 입력해주세요.', '', 'info', '닫기')  return false;  }  $('#phone1\_val').removeClass('border\_validate\_err');  $('#phone2\_val').removeClass('border\_validate\_err');  $('#phone3\_val').removeClass('border\_validate\_err');  return true;  }  if(this.fnValidate()){  .. 코드 생략..  }  .. 코드 생략.. |

2: 회원 정보를 입력한 후 하단의 [회원 가입] 버튼을 누르면, submitClick 함수가 실행된다.

58: fnValidate 함수는 입력 값의 유효성을 체크하는 함수인데, 하나라도 입력 값이 유효하지 않다면 false를 반환한다. 모든 입력 값이 유효하다면 line 59의 위치에서 아이디 중복 체크 api를 호출한다.

5~7: 정규 표현식을 사용해 숫자(pattern1), 문자(pattern2), 특수 문자(pattern3) 패턴 변수를 선언한다.

|  |
| --- |
| NOTE |
| 정규 표현식은 특정 규칙의 문자열을 표현하는 언어 형식이다. 문자열의 검색, 치환하는 용도로 사용하고 패턴을 사용하기 때문에 짧은 코드로 조건문을 구현할 수 있다. |

9~13: 비밀번호 컬럼의 입력 값이 없다면, 재입력 메시지를 표시하고 false를 반환한다.

16~20: 비밀번호 컬럼의 입력 값에 공백이 있다면, 공백제거 메시지를 표시한다. /\s/는 공백을 나타내는 정규식이다. search 함수를 사용해 str 변수에 공백이 몇 번째 위치에 존재하는지 체크 표시를 한다. 공백이 없다면 -1을 반환한다.

21~22: 비밀번호 컬럼의 입력 값에 숫자, 문자(영어), 특수 문자 중에 하나라도 값이 없거나 입력 값이 8~16자 사이가 아니라면 false를 반환한다.

35~40: 비밀번호와 비밀번호 확인 컬럼의 입력 값이 일치하지 않다면, false를 반환한다.

43~44: 핸드폰 컬럼의 입력 값이 하나라도 빈 값이라면, false를 반환한다.

특정 컬럼이 유효하지 않다면 해당 태그에 addClass('border\_validate\_err') 함수를 실행한다. 태그에 border\_validate\_err 클래스가 추가되면 테두리가 생기도록 css 파일에 정의돼 있다. 이와반대로 사용자가 정상적으로 재입력을 마쳤다면, removeClass('border\_validate\_err')를 실행해 테두리를 삭제한다.

|  |
| --- |
|  |

**<그림 217>**

150. REACT 회원 가입 페이지 만들기 3 - 아이디 중복 체크하기

**• 학습 내용:** 입력한 이메일로 기존에 가입된 계정이 있는지 확인한다.

**• 힌트 내용:** 이미 가입된 계정이라면, 사용자에게 이미 계정이 존재한다는 메시지를 표시한다.

회원 정보에서 아이디는 유일한 값이어야 한다. 회원 정보를 삽입하기 전 중복 체크를 위한 node api를 호출하고 db에 동일한 계정이 존재하는지 확인한다.

Register 폴더의 Register.js 파일을 확인한다.

<소스> Register.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24 | .. 코드 생략..  if(this.fnValidate()){  this.state.full\_email = this.email\_val\_checker+'@'+this.email2\_val\_checker  axios.post('/api/register?type=dplicheck', {  is\_Email: this.email\_val\_checker+'@'+this.email2\_val\_checker  })  .then( response => {  try {  const dupli\_count = response.data.json[0].num;  if(dupli\_count !== 0){  $('#email\_val').addClass('border\_validate\_err');  $('#email2\_val').addClass('border\_validate\_err');  this.sweetalert('이미 존재하는 이메일입니다.', '', 'info', '닫기')  }else{  $('#email\_val').removeClass('border\_validate\_err');  $('#email2\_val').removeClass('border\_validate\_err');  this.fnSignInsert('signup', e)  }  } catch (error) {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기')  }  })  .catch( response => { return false; } );  .. 코드 생략.. |

2: 모든 입력 값이 유효성 검사를 통화하면, fnValidate 함수는 true를 반환한다.

4~6: 중복 체크 node api(/api/register?type=dplicheck)를 호출하면서 파라미터로 입력받은 이메일 컬럼의 값을 전달한다

9~18: 중복 체크 node api는 중복된 계정의 수(response.data.json[0].num)를 반환한다. 중복된 계정이 있다면 line 13의 메시지를 표시한다. 새로운 계정이라면 회원 정보 삽입 node api를 호출하는 line 17의 fnSignInsert 함수를 실행한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19 | .. 코드 생략..  else if(type == "dplicheck"){  //이메일 중복 체크  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');    //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'select';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'selectUserDpliCheck';  router.use('/', dbconnect\_Module);  next('route')  } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: type 파라미터의 값이 dplicheck일 경우, 중복 체크 mapper 정보를 req.body에 할당한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10 | .. 코드 생략..  <select id="selectUserDpliCheck">  SELECT  count(\*) as num  FROM  react.react\_user  WHERE useremail = #{is\_Email}  </select>  .. 코드 생략.. |

3~8: react\_user 테이블에 파라미터로 전달받은 이메일(is\_Email) 데이터가 존재하는지 조회한다. 조회된 데이터의 개수를 count 함수로 계산한다.

|  |
| --- |
|  |

**<그림 218>**

151. REACT 회원 가입 페이지 만들기 3 - 회원 가입 api 호출하기

**• 학습 내용:** 입력한 이메일로 기존에 가입된 계정이 있는지 확인한다.

**• 힌트 내용:** 이미 가입된 계정이라면, 사용자에게 이미 계정이 존재한다는 메시지를 표시한다.

회원 정보에서 아이디는 유일한 값이어야 한다. 회원 정보를 삽입하기 전 중복 체크를 위한 node api를 호출하고 db에 동일한 계정이 존재하는지 확인한다.

Register 폴더의 Register.js 파일을 확인한다.

<소스> Register.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24 25  26 27 | .. 코드 생략..  this.fnSignInsert = async (type, e) => {  var jsonstr = $("form[name='frm']").serialize();  jsonstr = decodeURIComponent(jsonstr);  var Json\_form = JSON.stringify(jsonstr).replace(/\"/gi,'')  Json\_form = "{\"" +Json\_form.replace(/\&/g,'\",\"').replace(/=/gi,'\":"')+"\"}";    try {  const response = await fetch('/api/register?type='+type, {  method: 'POST',  headers: {  'Content-Type': 'application/json',  },  body: Json\_form,  });  const body = await response.text();  if(body === "succ"){  this.sweetalert('회원 가입이 완료됐습니다.', '', 'info', '닫기')  this.props.history.push('/');  }else{  this.sweetalert('작업 중 에러가 발생했습니다.', body, 'error', '닫기');  }  } catch (error) {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기');  }  }  .. 코드 생략.. |

2: 유효성 검사와 계정 중복 체크가 완료되면, fnSignInsert 함수를 실행한다.

3~4: <form> 태그에 있는 데이터를 한꺼번에 전송하기 위해 <form> 태그의 데이터들을 serialize 함수로 인코딩해 문자열을 만든다. 이때 한글이 인코딩돼 알아볼 수 없게 되는데, decodeURIComponent 함수를 사용해 디코딩하면 다시 한글로 변환된다.

5~6: form 데이터를 json 형태의 문자열로 가공한다.

9~15: 회원 정보 삽입 node api(/api/register?type= signup)를 post 방식으로 호출하고 body에 form 데이터를 가공한 Json\_form 변숫값을 할당해 전달한다.

16~22: 정상적으로 회원 정보를 삽입해 response로 succ를 반환받았다면, 회원 가입 완료 메시지를 표시한다. 그리고 line 19의 history.push 함수를 사용해 로그인 화면(/ 경로)으로 이동한다.

|  |
| --- |
|  |

**<그림 219>**

152. NODE 로그인 Api 만들기 1 - 라우터 분기, 쿼리 추가

**• 학습 내용:** 로그인 정보를 파라미터로 전달받아 조회 쿼리를 실행한다.

**• 힌트 내용:** 회원 아이디로 select 쿼리를 실행해 암호화된 비밀번호를 조회할 수 있다.

로그인 api를 호출할 때 필수적으로 아이디와 비밀번호를 파라미터로 전달한다. 전달받은 비밀번호는 암호화되지 않았기 때문에 db에 저장된 회원 비밀번호와 비교할 수 없다. 비교를 위해 아이디만으로 쿼리를 조회한다. 그리고 파라미터로 전달된 비밀번호를 암호화해서 db에서 조회된 비밀번호와 비교해야 한다.

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10 | .. 코드 생략..  var usersRouter = require("./routes/UsersRout");  var app = express();  .. 코드 생략..  app.use("/api/register", usersRouter);  app.use("/api/LoginForm", usersRouter);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

7: /api/LoginForm 경로로 api가 호출되면, line 2에서 지정한 회원 정보 라우터 파일 경로로 라우팅한다. line 6에서 사용한 회원 가입 api와 동일한 라우터 파일을 사용한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 | .. 코드 생략..  }else if(type == "signin"){  //로그인 조회  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'select';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'selectLoginCheck';  router.use('/', dbconnect\_Module);  next('route')    } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: type 파라미터의 값이 signin일 경우, 로그인 정보를 select하는 mapper 정보를 req.body에 할당한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16 | .. 코드 생략..  <select id="selectLoginCheck">  SELECT  username  , userorg  , useremail  , userpassword  , usermajor  , userphone  , userflag  FROM  react.react\_user  WHERE useremail = #{is\_Email}  </select>  .. 코드 생략.. |

14: 파라미터로 전달된 아이디인 is\_Email 변수를 useremail 컬럼과 비교해 일치하는 데이터를 조회한다. useremail 컬럼은 react\_user 테이블의 기본 키이기 때문에 1개의 데이터가 조회된다.

8: 회원 가입을 할 때 bcrypt로 암호화해 저장했던 userpassword(비밀번호) 컬럼이 조회된다.

153. NODE 로그인 Api 만들기 2 - bcrypt로 비밀번호 비교하기

**• 학습 내용:** bcrypt를 사용해 파라미터로 받은 비밀번호와 db에서 조회한 비밀번호를 비교한다.

**• 힌트 내용:** 파라미터로 전달받은 비밀번호를 bcrypt로 암호화해 비교한다.

회원 가입 api에서는 bcrypt의 hash 함수를 사용해 비밀번호를 암호화해서 db에 저장했다. bcrypt의 compare 함수를 사용하면, 회원 가입에 사용한 암호화 방식과 동일한 방법을 사용한다. 파라미터로 전달받은 비밀번호를 같은 방식으로 암호화해 db에서 조회한 비밀번호와 비교한다.

routes 폴더의 dbconnect\_Module.js 파일을 확인한다.

<소스> dbconnect\_Module.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41 | .. 코드 생략..  connection.query(query, function (error, results) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  if(results != undefined){  string = JSON.stringify(results);  var json = JSON.parse(string);  if (req.body.crud == "select") {  if (param.mapper\_id == "selectLoginCheck") {  if (json[0] == undefined) {  res.send(null);  } else {  bcrypt.compare(req.body.is\_Password, json[0].userpassword, function(  err,  login\_flag  ) {  if (login\_flag == true) {  res.send({ json });  } else {  res.send(null);  }  });  }  } else {  res.send({ json });  }  }else{  res.send("succ");  }  }else{  res.send("error");  }  connection.release();  console.log("========= Node Mybatis Query Log End =========\n");  });.. 코드 생략.. |

14~29: select 쿼리 중 mapper\_id가 selectLoginCheck인 경우 비밀번호 비교해 response를 전달한다.

15~17: selectLoginCheck 쿼리의 실행 결과 파라미터로 전달받은 아이디와 일치하는 계정이 db에 없을 경우, null을 response로 전달한다.

18~20: bcrypt의 compare 함수로 파라미터로 전달받은 비밀번호(req.body.is\_Password)와 db에서 조회한 암호화된 비밀번호(json[0].userpassword)를 비교한다. 두 비밀번호가 일치한다면 line 20의 login\_flag 변숫값이 true로, 일치하지 않는다면 false가 반환된다.

22~26: 비밀번호가 일치했다면 조회된 데이터(json)를 line 23에서 response로 전달한다. 일치하지 않는다면 line 25에서 null을 response로 전달한다.

154. REACT 로그인 페이지 만들기 1 - 로그인 api 호출하기

**• 학습 내용:** 로그인 정보를 입력받고 로그인 api를 호출한다.

**• 힌트 내용:** api 호출 시 request에 아이디와 비밀번호를 전달한다.

로그인 node api 호출 시 쿼리에 필요한 변수들을 axios의 post 함수 파라미터로 전달한다. 로그인 아이디와 비밀번호를 입력받아 json 형태로 할당해 request로 전달한다.

components 폴더의 LoginForm.js 파일을 확인한다.

<소스> LoginForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50  51 52 53  54 | .. 코드 생략..  submitClick = (e) => {  this.email\_val = $('#email\_val').val();  this.pwd\_val = $('#pwd\_val').val();  if(this.email\_val === '' || this.pwd\_val === ''){  this.sweetalert('이메일과 비밀번호를 확인해주세요.', '', 'info', '닫기')  }else{  axios.post('/api/LoginForm?type=signin', {  is\_Email: this.email\_val,  is\_Password: this.pwd\_val  })  .then( response => {  var userid = response.data.json[0].useremail  var username = response.data.json[0].username  var upw = response.data.json[0].userpassword    if(userid != null && userid != ''){  this.sweetalert('로그인 됐습니다.', '', 'info', '닫기')  .. 코드 생략..  setTimeout(function() {  window.location.href = '/SoftwareList';  }.bind(this),1000);  }else{  this.sweetalert('이메일과 비밀번호를 확인해주세요.', '', 'info', '닫기')  }  })  .catch( error => {this.sweetalert('이메일과 비밀번호를 확인해주세요.', '', 'info', '닫기')} );  }  }  .. 코드 생략..  <section className="main">  <div className="m\_login">  <h3><span><img src={require("../img/main/log\_img.png")} alt="" />  </span>LOGIN</h3>  <div className="log\_box">  <div className="in\_ty1">  <span><img src={require("../img/main/m\_log\_i3.png")} alt="" /></span>  <input type="text" id="email\_val" placeholder="이메일" />  </div>  <div className="in\_ty1">  <span className="ic\_2">  <img src={require("../img/main/m\_log\_i2.png")} alt="" />  </span>  <input type="password" id="pwd\_val" placeholder="비밀번호" />  </div>  <ul className="af">  <li><Link to={'/register'}>회원 가입</Link></li>  <li className="pwr\_b"><a href="#n">비밀번호 재설정</a></li>  </ul>  <div className="s\_bt" type="" onClick={(e) => this.submitClick(e)}>로그인</div>  </div>  </div>  </section> |

39: id가 email\_val인 text 타입의 <input> 태그로, 로그인 아이디를 입력받는다.

45: id가 pwd\_val인 password 타입의 <input> 태그로, 비밀번호를 입력받는다.

51: 아이디와 비밀번호를 입력하면 로그인 버튼을 클릭해 submitClick 함수를 실행한다.

5~8: 아이디나 비밀번호가 입력되지 않았다면, 재입력 메시지를 표시한다.

9~12: 로그인 api(/api/LoginForm?type=signin)를 호출하고 파라미터로 로그인 아이디(is\_Email)와 비밀번호(is\_Password)를 전달한다.

14~16: 로그인 api 호출 결과 반환된 response에서 아이디(useremail), 회원명(username), 암호화된 비밀번호(userpassword)를 각각 변수에 할당한다.

18~19: 일치하는 회원 정보가 db에 존재해 회원 정보가 정상적으로 반환됐다면, 로그인 성공 메시지를 표시한다.

21~23: 로그인이 정상적으로 완료되면, Software Tools 목록(/SoftwareList)으로 페이지를 이동한다. line 19의 로그인 성공 메시지를 1초 표시하고 페이지 이동을 하기 위해 setTimeout 함수를 사용한다.

정상적으로 로그인에 성공하면, 다음과 같이 메시지가 표시된다.

|  |
| --- |
|  |

**<그림 220>**

155. 회원 정보 암호화 Api 만들기 - jwt로 회원 정보 암호화하기

**• 학습 내용:** jsonwebtoken 패키지로 회원 정보를 암호화한다.

**• 힌트 내용:** 쿠키에 저장할 회원 정보를 암호화하기 위해 비밀 를 사용한다.

로그인 정보가 정상적으로 확인됐다면, 쿠키에 회원 정보를 저장해 로그인을 유지해야 한다. 이때 회원 정보가 그대로 쿠키에 저장되면 위험하기 때문에 jsonwebtoken 패키지를 사용해 암호화한다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save jsonwebtoken]를 입력하면 다음과 같이 npm이 jsonwebtoken를 설치한다.

|  |
| --- |
|  |

**<그림 221>**

node 경로(C:\react200) 안에 있는 ignorefile 폴더의 jwt.js 파일을 확인한다.

<소스> jwt.js

|  |  |
| --- | --- |
| 1 2  3 | let jwtObj = {};  jwtObj.secret = "react200"  module.exports = jwtObj |

1~2: json 객체(jwtObj)를 생성해 key(secret)와 value(react200)를 할당한다. value 값은 비밀 키가 되는데 텍스트로, 임의의 값으로 정하고 외부에 표시되지 않도록 관리한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19 20  21  22  23  24 | .. 코드 생략..  let jwt = require("jsonwebtoken");  let secretObj = require("../ignorefile/jwt");  .. 코드 생략..  }else if(type == "SessionState"){  var userid = req.body.is\_Email  var name = req.body.is\_UserName  try {  let token1 = jwt.sign(  { email: userid },  secretObj.secret,  { expiresIn: '60m' })    let token2 = jwt.sign(  { username: name },  secretObj.secret,  { expiresIn: '60m' })  res.send({"token1":token1, "token2":token2});  } catch (error) {  res.send(error)  }  }  .. 코드 생략.. |

2~3: jsonwebtoken 패키지를 require해 사용할 수 있도록 한다. 비밀 키가 저장된 파일(/ignorefile/jwt.js)의 json 객체를 secretObj 변수에 할당한다.

5~8: type 파라미터의 값이 SessionState일 경우, 파라미터로 전달받은 아이디와 회원명을 각각 변수에 할당한다.

10~13: jsonwebtoken 패키지의 sign 함수를 사용해 아이디의 암호화된 토큰을 생성한다. Email 토큰 변수에 파라미터로 전달받은 아이디(userid)를 할당한다. 비밀 키(secretObj.secret)를 sign 함수의 두 번째 파라미터로 호출한다. expiresIn 속성으로 토큰이 유효한 시간을 60분으로 할당한다.

|  |
| --- |
| NOTE |
| Jsonwebtoken은 헤더(header), 내용(payload), 서명(signature)이 마침표(.)로 연결돼 있는 구조다. 헤더에는 토큰 유형(jwt)과 서명에 사용하는 암호화 알고리즘 정보가 있다. 내용에는 실제로 저장되는 회원 아이디와 같은 정보가 저장된다. 서명에는 비밀 키로 암호화된 값이 할당되는데, 내용에 저장된 데이터가 필요한 시점에 다시 서명을 비밀 키로 복호화해 인증한다. 인증이 정상적으로 완료되면, 내용에 저장된 데이터를 사용할 수 있다. |

15~18: 회원명도 아이디와 동일한 방식으로 토큰을 만든다.

19: 가공된 토큰을 json 형태의 데이터로 response에 반환한다.

156. REACT 쿠키로 로그인 유지하기 - 쿠키에 회원 정보 저장하기

**• 학습 내용:** react-cookies 패키지를 사용해 암호화된 회원 정보를 쿠키에 저장한다.

**• 힌트 내용:** jwt로 암호화된 회원 정보를 react-cookies의 save 함수로 쿠키에 저장한다.

로그인 api 호출를 호출해 쿠키에 저장할 데이터(아이디, 회원명, 암호화된 비밀번호)가 반환됐다. 아이디와 회원명은 jwt로 암호화했다. bcypt로 암호화된 비밀번호는 유출돼도 로그인에 사용할 수 없기 때문에 그대로 사용해도 된다. 세 가지 데이터를 react-cookies의 save 함수로 쿠키에 저장한다.

cmd 창을 열어 client 폴더 경로로 이동한다. [npm install react-cookies --save]를 입력하면 다음과 같이 npm이 react-cookies를 설치한다.

|  |
| --- |
|  |

**<그림 223>**

components 폴더의 LoginForm.js 파일을 확인한다.

<소스> LoginForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40 | .. 코드 생략..  axios.post('/api/LoginForm?type=signin', {  is\_Email: this.email\_val,  is\_Password: this.pwd\_val  })  .then( response => {  var userid = response.data.json[0].useremail  var username = response.data.json[0].username  var upw = response.data.json[0].userpassword    if(userid != null && userid != ''){  this.sweetalert('로그인 됐습니다.', '', 'info', '닫기')  const expires = new Date()  expires.setMinutes(expires.getMinutes() + 60)    axios.post('/api/LoginForm?type=SessionState', {  is\_Email: userid,  is\_UserName: username,  })  .then( response => {  cookie.save('userid', response.data.token1  , { path: '/', expires })  cookie.save('username', response.data.token2  , { path: '/', expires })  cookie.save('userpassword', upw  , { path: '/', expires })  })  .catch( error => {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기');  });    setTimeout(function() {  window.location.href = '/SoftwareList';  }.bind(this),1000);  }else{  this.sweetalert('이메일과 비밀번호를 확인해주세요.', '', 'info', '닫기')  }  })  .. 코드 생략.. |

14~15: 현재 시간 정보에 60분을 더한 값을 expires 변수에 할당한다.

22: jwt로 암호화된 아이디(response.data.token1) 값을 react-cookies 패키지의 save 함수로 쿠키에 저장한다.

23: 쿠키 유지 시간을 line 15에서 지정한 expires 변수(60분)로 할당한다.

24~25: 회원명도 아이디와 동일한 방식으로 쿠키에 저장된다.

26~27: 암호화된 비밀번호는 그대로 쿠키에 저장된다.

로그인 정보가 정상적으로 조회되고 쿠키에 회원 정보가 저장되면 다음과 같이 Software Tools 목록 페이지로 이동한다. [개발자 도구] - [Application 탭] - [Storage] - [Cookies]를 선택한 후에 <http://localhost:3000>을 클릭하면 쿠키의 name과 value 값을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 222>**

157. 회원 정보 복호화 Api 만들기 - jwt로 회원 정보 복호화하기

**• 학습 내용:** jsonwebtoken 패키지로 회원 정보를 복호화한다.

**• 힌트 내용:** 쿠키에서 불러온 회원 정보를 복호화 하기 위해 비밀 를 사용한다.

보안상의 이유로 암호화해 쿠키에 저장했던, 회원 정보를 사용해야 할 때가 있다. 화면에 회원 정보를 표시해야 하거나 로그인 인증이 다시 필요한 시점이다. 회원 정보의 복호화에는 암호화를 할 때 사용했던 비밀 키가 사용된다.

node 경로(C:\react200) 안에 있는 ignorefile 폴더의 jwt.js 파일을 확인한다.

<소스> jwt.js

|  |  |
| --- | --- |
| 1 2  3 | let jwtObj = {};  jwtObj.secret = "react200"  module.exports = jwtObj |

2: 암호화할 때 사용한 비밀 를 그대로 사용한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19 20  21  22 | .. 코드 생략..  let jwt = require("jsonwebtoken");  let secretObj = require("../ignorefile/jwt");  .. 코드 생략..  }else if(type == "SessionConfirm"){  try {  let token1 = req.body.token1;  let token2 = req.body.token2;    if(token1 != undefined && token1 != '' & token2 != undefined && token2 != ''){  let decoded1 = jwt.verify(token1, secretObj.secret);  let decoded2 = jwt.verify(token2, secretObj.secret);  res.send({"token1":decoded1.email, "token2":decoded2.username});  }else{  res.send({"token1":"", "token2":""});  }  } catch (error) {  res.send(error)  }  }  .. 코드 생략.. |

2~3: jsonwebtoken 패키지를 require해 사용할 수 있도록 한다. 비밀 키가 저장된 파일(/ignorefile/jwt.js)의 json 객체를 secretObj 변수에 할당한다.

5~9: type 파라미터의 값이 SessionConfirm일 경우, 파라미터로 전달받은 아이디와 회원명을 각각 변수에 할당한다. 이때 두 변수는 암호화된 상태다.

11~13: 회원 정보가 존재할 경우, jsonwebtoken 패키지의 verify 함수를 사용해 회원 정보를 복호화한다. 암호화된 회원 정보(token1, token2)를 비밀 키(secretObj.secret)를 사용해 복호화하고 각각 변수(decoded1, decoded2)에 할당한다.

14: 복호화된 이메일(decoded1.email)과 회원명(decoded2.username)을 json 형태로 response에 담아 전달한다.

16: 복호화 과정에서 에러가 발생했다면, 빈 값을 전달한다.

158. 로그인 회원에게 권한 부여하기 - 쿠키에서 회원 정보 불러오기

**• 학습 내용:** react-cookies 패키지를 사용해 암호화된 회원 정보를 불러온다.

**• 힌트 내용:** 불러온 회원 정보로 재인증하고 인증된 경우 페이지 접근 권한을 부여한다.

쿠키에 저장됐던 데이터(아이디, 회원명, 암호화된 비밀번호)를 react-cookies의 load 함수로 불러온다. 불러온 아이디와 암호화된 비밀번호로 사용자 인증을 하고 인증이 완료된 경우에만 특정 페이지에 접근 권한을 부여한다.

components 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43  44  45  46 | .. 코드 생략..  componentDidMount() {  axios.post('/api/LoginForm?type=SessionConfirm', {  token1 : cookie.load('userid')  , token2 : cookie.load('username')  })  .then( response => {  this.state.userid = response.data.token1  let password = cookie.load('userpassword')  if(password !== undefined){  axios.post('/api/LoginForm?type=SessionSignin', {  is\_Email: this.state.userid,  is\_Token : password  })  .then( response => {  if(response.data.json[0].useremail === undefined){  this.noPermission()  }  })  .catch( error => {  this.noPermission()  });  }else{  this.noPermission()  }  })  .catch( response => this.noPermission());  }  noPermission = (e) => {  if(window.location.hash != 'nocookie'){  this.remove\_cookie();  window.location.href = '/login/#nocookie';  }  };  remove\_cookie = (e) => {  cookie.remove('userid', { path: '/'});  cookie.remove('username', { path: '/'});  cookie.remove('userpassword', { path: '/'});  }  .. 코드 생략..  <Route exact path='/' component={LoginForm} />  <Route path='/login' component={LoginForm} />  .. 코드 생략.. |

App.js는 react 페이지 라우터다. 그래서 페이지를 이동으로 새로 고침이 발생할 때마다 코드가 실행된다.

2: 생명주기 함수 실행 순서대로 render 함수가 실행되고 componentDidMount 함수가 실행된다.

3~6: 회원 정보 복호화 api(/api/LoginForm?type=SessionConfirm)를 호출한다. 이때 쿠키에 있는 아이디와 회원명을 load 함수로 불러와 파라미터로 전달한다.

9~10: 복호화된 아이디와 쿠키에 저장돼 있던 암호화된 비밀번호를 각각 변수에 할당한다.

12~15: 회원 정보 재인증을 위해 node api(/api/LoginForm?type=SessionSignin)를 호출한다. 파라미터로 아이디와 암호화된 비밀번호를 전달한다.

18, 22, 25, 28: 쿠키에 비밀번호가 없거나 회원 정보가 일치하지 않거나 에러가 발생한 경우 noPermission 함수를 실행한다.

38~42: cookie.remove 함수를 사용해 쿠키에서 회원 정보를 삭제하는 remove\_cookie 함수를 정의한다.

34~36: 회원 정보가 유효하지 않은 경우, remove\_cookie 함수를 실행해 모든 쿠를 지운다. 그리고/login/#nocookie 경로로 페이지 이동을 한다.

32: line 34에서 호출한 페이지 경로에 hash(/#nocookie)가 있다. 회원 정보가 유효하지 않는 경우 계속 window.location.href 함수가 실행되고 무한 새로 고침 현상이 발생한다. url에 hash(/#nocookie) 값이 있는 경우, 페이지 이동을 하지 않도록 하면 무한 로딩을 막을 수 있다.

45: line 34에서 호출한 /login 경로는 로그인 페이지(LoginForm)이다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | .. 코드 생략..  <select id="selectSessionLoginCheck">  SELECT  username  , userorg  , useremail  , userpassword  , usermajor  , userphone  , userflag  FROM  react.react\_user  WHERE useremail = #{is\_Email}  AND userpassword = #{is\_Token}  </select>  .. 코드 생략.. |

2~16: /api/LoginForm?type=SessionSignin 경로로 node api를 호출하면, selectSessionLoginCheck 쿼리가 실행된다. line 15에서 암호화된 비밀번호를 비교한다.

로그인이 되지 않은 상태에서 <http://localhost:3000/SoftwareList> 경로를 호출하면, 강제로 로그인 페이지(/login)로 이동시킨다.

|  |
| --- |
|  |

**<그림 224>**

159. 로그인 세션 시간 관리 - 쿠키 유효 시간 연장하기

**• 학습 내용:** 페이지 새로 고침을 할 때마다 쿠키 유효 시간을 연장한다.

**• 힌트 내용:** load 함수로 쿠키 정보를 불러와 save 함수로 다시 쿠를 저장한다.

로그인을 성공하면 쿠키 시간이 60분 할당된다. 로그인 이후 사용자가 페이지 새로 고침을 하게 되면, 그 시점부터 쿠키 시간을 새로 60분 할당한다. 쿠키 시간 재할당 코드는 모든 페이지에서 표시되는 header에 구현한다.

Header 폴더의 Header admin.js 파일을 확인한다.

<소스> Header admin.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27 | .. 코드 생략..  componentDidMount() {  var cookie\_userid = cookie.load('userid')  var cookie\_usernm = cookie.load('username')  var cookie\_password = cookie.load('userpassword')  if(cookie\_userid != undefined){  const expires = new Date()  expires.setMinutes(expires.getMinutes() + 60)  cookie.save('userid', cookie\_userid  , { path: '/', expires })  cookie.save('username', cookie\_usernm  , { path: '/', expires })  cookie.save('userpassword', cookie\_password  , { path: '/', expires })  $('.menulist').show()  $('.hd\_top').show()  }else{  $('.menulist').hide()  $('.hd\_top').hide()  }  this.callSessionInfoApi()  }.. 코드 생략.. |

Header admin.js는 모든 페이지에서 새로 고침이 발생할 때마다 코드가 실행된다.

3~5: 쿠키에 저장된 회원 정보를 불러와 각각 변수에 할당한다.

8~17: 쿠키에 아이디 정보가 존재한다면, 현재 시간에 60분을 더해 쿠키 유효 시간을 할당한다. 쿠키 값은 변화 없이 유효 시간만 60분이 더해져 저장된다

160. 로그인 상태에서만 header 표시하기, 로그아웃 구현하기

**• 학습 내용:** 로그인 상태에서만 header 정보를 표시한다.

**• 힌트 내용:** header는 회원명, [로그아웃] 버튼, 페이지 내비게이션으로 구성된다.

로그인이 되지 않은 상태에서는 회원 정보와 [로그아웃] 버튼이 표시되면 안 된다. 내비게이션의 페이지 링크는 react-router-dom 패키지의 <Link> 태그로 돼 있다. <Link> 태그를 통해 페이지 이동을 할 경우, 페이지 새로 고침이 발생하지 않는다. 이 경우, App.js에 구현한 페이지 권한 제한 로직이 실행되지 않는다. 이런 이유에서 비로그인 상태에서는 페이지 내비게이션 영역을 표시하지 않는다.

Header 폴더의 Header admin.js 파일을 확인한다.

<소스> Header admin.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73 | .. 코드 생략..  componentDidMount() {  .. 코드 생략..  if(cookie\_userid != undefined){  .. 코드 생략..  $('.menulist').show()  $('.hd\_top').show()  }else{  $('.menulist').hide()  $('.hd\_top').hide()  }  this.callSessionInfoApi()  }  callSessionInfoApi = (type) => {  axios.post('/api/LoginForm?type=SessionConfirm', {  token1 : cookie.load('userid')  , token2 : cookie.load('username')  })  .then( response => {  this.setState({usernm : response.data.token2})  })  .catch( error => {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기');  });  }  myInfoHover () {  $(".hd\_left > li > .box1").stop().fadeIn(400);  }  myInfoLeave () {  $(".hd\_left > li > .box1").stop().fadeOut(400);  }  logout = async e => {  cookie.remove('userid', { path: '/'});  cookie.remove('username', { path: '/'});  cookie.remove('userpassword', { path: '/'});  window.location.href = '/login';  }  .. 코드 생략..  <div className="hd\_top">  <div className="top\_wrap ct1 af">  <ul className="hd\_left af">  <li className="my1" onMouseEnter={this.myInfoHover}  onMouseLeave={this.myInfoLeave}><b>내정보</b>  <div className="box0 box1">  <ul>  <li><a>내 정보 수정</a></li>  <li><a href="javascript:" onClick={this.logout}>로그아웃</a></li>  </ul>  </div>  </li>  <li className="my2"><b><span>0</span> 알림</b>  </li>  </ul>  <div className="hd\_right">  <p><span>'{this.state.usernm}'</span>님 반갑습니다.</p>  </div>  </div>  </div>  .. 코드 생략..  <ul className="af">  <li className="menulist">  <Link to={'/UserApproval'}> 사용자 관리</Link>  </li>  <li className="menulist">  <Link to={'/AdminResearchProject'}>Research Projects 관리</Link>  </li>  .. 코드 생략.. |

5~13: 쿠키에 회원 정보가 있을 경우, line 45의 <div> 태그 영역과 line 67, 70의 페이지 내비게이션 영역을 표시한다. 반대로 회원 정보가 없을 경우 표시하지 않는다.

14~28: 페이지에 표시할 회원명을 복호화 하기 위해 callSessionInfoApi 함수를 실행한다. 복호화 api(/api/LoginForm?type=SessionConfirm) 호출 결과 반환된 회원명(response.data.token2)을 state 변수(usernm)에 할당한다.

61: state 변수 usernm에 할당된 회원명이 표시된다.

48~49: 내 정보 영역에 마우스 커서를 올려놓으면 myInfoHover 함수가 실행되고 line 31에서 [로그아웃] 버튼을 표시한다. 이와 반대로 마우스 커서가 영역에서 벗어나면 myInfoLeave 함수가 실행되고 line 35에서 [로그아웃] 버튼이 사라진다.

38~43: line 53에서 [로그아웃] 버튼을 클릭하면, 쿠키에 저장된 회원 정보를 삭제하고 로그인 페이지(/login)로 이동한다.

로그인이 되지 않은 상태에서는 다음과 같이 header 영역이 표시되지 않는다.

|  |
| --- |
|  |

**<그림 225>**

로그인 이후 header가 표시되고 내 정보 영역을 클릭하면, [로그아웃] 버튼이 표시된다.

|  |
| --- |
|  |

**<그림 226>**

161. 비밀번호 재설정 메일 템플릿 만들기

**• 학습 내용:** 비밀번호 재설정 메일에 사용할 html 템플릿을 만든다.

**• 힌트 내용:** 템플릿에는 인증 후 웹 사이트로 다시 돌아올 수 있는 링크가 있어야 한다.

로그인 비밀번호를 잊어버린 경우, 웹 사이트의 아이디로 사용되는 이메일 인증을 통해 비밀번호를 재설정할 수 있다. 이메일 템플릿은 사용자가 이메일로 받게 되는 내용인데, html를 사용하면 웹 사이트 링크를 걸 수 있는 태그를 사용할 수 있다.

[Routes] - [template] 폴더의 mail\_template\_pw.html 파일을 확인한다.

<소스> mail\_template\_pw.html

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13 | .. 코드 생략..  <div align="center">  <a href="{replacetext}" target="\_blank" style="display: inline-block;  .. 코드 생략..  <span style="line-height:150%;">  <strong>  <span style="font-size: 30px; line-height: 45px;">비밀번호 변경하기</span>  </strong>  </span>  </a>  </div>  .. 코드 생략.. |

3: 메일 발송 api에서 mail\_template\_pw.html 파일을 읽고 {replacetext} 문자열을 찾아 웹 사이트 링크(http://localhost:3000/PwChangeForm/이메일/토큰)로 치환한다.

5: 이메일로 전송하는 html 파일의 경우, 스타일을 적용하려는 태그 안에 style 속성을 직접 사용해야 한다. 페이지 상단에 <style> 태그를 추가하거나 css 파일을 생성하면 스타일이 적용되지 않는다.

8: [비밀번호 변경하기] 버튼을 생성한다. 버튼을 누르면 line 3에서 href 속성 값으로 치환된 웹 사이트 링크로 이동한다.

|  |
| --- |
|  |

**<그림 227>**

162. 메일 발송 Api 만들기 - nodemailer 패키지 사용

**• 학습 내용:** nodemailer 패키지를 사용해 비밀번호 재설정 메일을 전송한다.

**• 힌트 내용:** gmail 계정을 연동하면, 하루 500개의 메일을 무료로 발송할 수 있다.

메일 발송 api를 호출할 때 파라미터로 메일 제목, 이메일(아이디), 비밀번호를 전달한다. 메일 템플릿의 [비밀번호 변경하기] 버튼 링크를 웹 사이트 주소에 이메일(아이디)과 비밀번호 토큰을 넣은 값으로 치환한다. 연동한 gmail 계정으로 이메일(아이디)에 메일을 발송하면, 송신자는 gmail 계정이 된다. 메일 템플릿 파일을 불러오기 위해 fs 패키지와 메일 발송을 위해 nodemailer 패키지를 설치한다.

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save nodemailer]를 입력하면 다음과 같이 npm이 nodemailer를 설치한다.

|  |
| --- |
|  |

**<그림 228>**

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install -- save fs]를 입력하면 다음과 같이 npm이 fs를 설치한다.

|  |
| --- |
|  |

**<그림 229>**

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | .. 코드 생략..  var MailRout = require("./routes/MailRout");  var app = express();  .. 코드 생략..  app.use("/api/mail", MailRout);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

7: /api/mail 경로로 api가 호출되면, line 2에서 지정한 메일 발송 라우터 파일 경로로 라우팅한다.

routes 폴더의 MailRout.js 파일을 확인한다.

<소스> MailRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40 41  42 43  44  45  46  47  48  49  50 | .. 코드 생략..  const nodemailer = require('nodemailer');  var fs = require('fs')  .. 코드 생략..  router.post('/', (req, res, next) => {  let email = req.body.is\_Email;  let subject = req.body.is\_Subject;  var password = req.body.is\_Password;  password = password.substr(0, 20)    let transporter = nodemailer.createTransport({  service: 'gmail',  host: 'smtp.gmail.com',  port: 465,  secure: true,  auth: {  user: '010\*\*\*\*\*\*\*\*a@gmail.com',  pass: '\*\*\*\*\*\*\*\*\*\*'  }  });  home\_url = 'http://localhost:3000'  var toHtml = ''  fs.readFile(\_\_dirname+'/template/mail\_template\_pw.html', function (err, html) {  toHtml = html.toString()  toHtml = toHtml.replace('{replacetext}', home\_url+'/PwChangeForm/'+ email +'/'+password)  })  setTimeout(function() {  let mailOptions = {  from: '010\*\*\*\*\*\*\*\*a@gmail.com',  to: email,  subject: subject,  html : toHtml  };  transporter.sendMail(mailOptions, function(error, info){  if (error) {  console.log(error);  }  else {  console.log('Email sent: ' + info.response);  }  });  res.send("succ");  }.bind(this),1000  );  });  module.exports = router; |

2~3: nodemailer와 fs 패키지를 require해 사용할 수 있도록 한다.

6~9: 이메일, 메일 제목, 암호화된 비밀번호를 파라미터로 전달받아 각각 변수에 할당한다.

10: 암호화된 비밀번호를 앞에서 20자만 잘라 password 변수에 할당한다. 20자의 password 변수는 이메일 인증을 위한 토큰으로 사용된다.

12~21: nodemailer 패키지의 createTransport 함수로 메일 서버의 정보를 세팅한다.

13~15: gmail 자원을 사용하기 위해 파라미터 변수를 할당한다. 이메일을 보내기 위해 smtp 프로토콜을 사용한다. smtp 프로토콜을 465 포트를 사용한다.

16: secure가 true면 smtp로 메일을 발송할 때 ssl을 사용해 암호화된 통신을 한다.

17~20: gmail 계정의 메일 주소를 user 변수에 비밀번호를 pass 변수에 할당한다.

25: fs 패키지의 readFile 함수를 사용해 메일 템플릿 html 파일을 불러온다. 콜백 함수에서 불러온 파일을 html 변수로 반환한다

26~27: html 파일을 문자열로 변환한다. 메일 템플릿에서 <a> 태그의 href 속성에 할당된 {replacetext} 문자열을 웹 사이트 경로로 치환한다. 로컬 서버의 경우 http://localhost:3000/PwChangeForm/\*\*\*\*@naver.com/$2b$10$hF58DTwP가2kX5와 같은 형태로 웹 사이트 경로가 생성된다. 외부 서버에서 서버를 구동하는 경우, line 23의 home\_url 값을 외부 서버 ip로 변경해야 한다.

30~46: html 파일을 불러와 문자열을 치환하는 데 일정 시간이 소요된다. 전송할 메일 템플릿 html 파일이 완전히 가공된 이후에 메일 발송을 해야 하기 때문에 1초의 지연 시간을 설정한다.

31~36: 메일 송신 주소(from), 메일 수신 주소(to), 메일 제목(subject), 메일 내용(html)을 mailOptions 변수에 할당한다.

37~44: line 12에서 생성한 transporter 객체를 사용해 이메일 발송 함수 sendMail을 실행한다. 이때 line 36에서 할당한 mailOptions 변수가 파라미터로 전달된다. 메일 발송의 결과를 로그로 출력한다.

45: 메일이 정상적으로 발송되면, succ 문자열을 response로 전달한다.

163. 비밀번호 조회 Api 만들기 - 라우터 분기, 쿼리 추가

**• 학습 내용:** 아이디와 회원명으로 암호화된 비밀번호를 조회한다.

**• 힌트 내용:** 암호화된 비밀번호는 유출돼도 로그인에 사용될 수 없다.

비밀번호를 잊은 상태이기 때문에 아이디와 비밀번호로 사용자 정보를 조회할 수 없다. 아이디와 회원명으로, 비밀번호 재설정에 필요한 암호화된 비밀번호를 조회한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 | .. 코드 생략..  }else if(type == "pwreset"){  //비밀번호 재설정시 이메일과 이름으로 회원 정보 조회  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'select';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'selectLoginResetCheck';  router.use('/', dbconnect\_Module);  next('route')    } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: type 파라미터의 값이 pwreset일 경우, 암호화된 비밀번호를 select하는 mapper 정보를 req.body에 할당한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11 | .. 코드 생략..  <select id="selectLoginResetCheck">  SELECT  userpassword  FROM  react.react\_user  WHERE useremail = #{is\_Email}  AND username = #{is\_Name}  </select>  .. 코드 생략.. |

3~9: 아이디(is\_Email)와 회원명(is\_Name)으로 회원 정보 테이블(react\_user)을 조회해 암호화된 비밀번호(userpassword)를 조회한다.

164. 아이디와 회원명으로 비밀번호 조회 Api 호출하기

**• 학습 내용:** 비밀번호를 잊어버린 경우, 아이디와 회원명으로 비밀번호를 조회한다.

**• 힌트 내용:** 조회한 회원 정보 중 암호화된 비밀번호가 이메일 인증의 토큰으로 사용된다.

비밀번호 재설정 메일을 발송하기 위해 비밀번호가 아닌 값으로 회원 정보를 조회해야 한다. 아이디(메일 주소)와 회원명을 조건으로, 암호화된 비밀번호를 조회한다. 메일 발송 함수를 호출하면서 파라미터로 아이디(메일 주소)와 암호화된 비밀번호를 전달한다.

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9 | .. 코드 생략..  var MailRout = require("./routes/MailRout");  var app = express();  .. 코드 생략..  app.use("/api/mail", MailRout);  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

7: /api/mail 경로로 api가 호출되면, line 2에서 지정한 메일 발송 라우터 파일 경로로 라우팅한다.

components 폴더의 LoginForm.js 파일을 확인한다.

<소스> LoginForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40 41  42 43  44  45  46  47  48  49  50 51  52 53  54  55  56  57  58  59  60 61  62 63  64  65  66  67  68  69  70  71  72 | .. 코드 생략..  pwdResetClick = () => {  $('.signin').hide();  $('.chgpw').fadeIn();  $('.chgpw').css('display','table-cell');  }  pwdResetCancleClick = () => {  $('.chgpw').hide();  $('.signin').fadeIn();  $('.signin').css('display','table-cell');  }  pwdResetConfim = (e) => {  this.reset\_email = $('#reset\_email\_val').val();  this.reset\_name = $('#reset\_name\_val').val();  if(this.reset\_email === '' || this.reset\_name === ''){  this.sweetalert('이메일과 성명을 확인해주세요.', '', 'info', '닫기')  }else{  axios.post('/api/LoginForm?type=pwreset', {  is\_Email: this.reset\_email,  is\_Name: this.reset\_name,  })  .then( response => {  var userpassword = response.data.json[0].userpassword  userpassword = userpassword.replace(/\//gi,"가")    if(userpassword != null && userpassword != ''){  this.sendEmail(this.reset\_email, 'react200 비밀번호 재설정 메일', userpassword)  }else{  this.sweetalert('이메일과 성명을 확인해주세요.', '', 'info', '닫기')  }  })  .catch( error => {  this.sweetalert('이메일과 성명을 확인해주세요.', '', 'info', '닫기')  });  }  }  .. 코드 생략..  </span>LOGIN</h3>  .. 코드 생략..  </div>  <ul className="af">  <li><Link to={'/register'}>회원 가입</Link></li>  <li className="pwr\_b" onClick={this.pwdResetClick}><a href="#n">비밀번호 재설정</a></li>  </ul>  <div className="s\_bt" type="" onClick={(e) => this.submitClick(e)}>로그인</div>  </div>  </div>  <div className="m\_login m\_pw chgpw">  <h3 className="pw\_ls">비밀번호 재설정 <span className="compl1">완료</span></h3>  <div className="log\_box">  <div className="pw\_one">  <div className="in\_ty1">  <span><img src={require("../img/main/m\_log\_i3.png")} alt="" /></span>  <input type="text" id="reset\_email\_val" name="" placeholder="이메일"/>  </div>  <div className="in\_ty1">  <span className=""><img src={require("../img/main/m\_log\_i1.png")} alt="" /></span>  <input type="text" id="reset\_name\_val" name="" placeholder="성명"/>  </div>  <div className="btn\_confirm btn\_confirm\_m">  <div className="bt\_ty bt\_ty\_m bt\_ty1 cancel\_ty1"  onClick={this.pwdResetCancleClick}>취소</div>  <a href="#n" className="bt\_ty bt\_ty\_m bt\_ty2 submit\_ty1"  onClick={this.pwdResetConfim}>확인</a>  </div>  </div>  </div>  .. 코드 생략.. |

46: 로그인 화면에서 [비밀번호 재설정] 버튼을 누르면, line 2의 pwdResetClick 함수가 실행된다. 이때 로그인 폼이 사라지고 비밀번호 재설정 폼이 표시된다.

64~65: 비밀번호 재설정 폼이 표시된 상태에서 [취소] 버튼을 누르면 line 9의 pwdResetCancleClick 함수가 실행된다. 이때 pwdResetClick 함수와 반대로 비밀번호 재설정 폼이 사라지고 로그인 폼이 표시된다.

66~67: 57, line 61에서 이메일(아이디)와 성명(회원명)을 입력하고 [확인] 버튼을 누르면 pwdResetConfim 함수가 실행된다.

16~19: pwdResetConfim 함수가 실행되면 line 57, 61의 <input> 태그에 id로 접근하고 입력 값을 가져온다. 이메일과 성명이 입력되지 않았다면 알림 메시지를 표시한다.

21~24: 비밀번호 조회 node api(/api/LoginForm?type=pwreset)를 호출하면서 파라미터로 아이디(is\_Email)와 회원명(is\_Name)을 전달한다.

26~27: 조회된 암호화된 비밀번호를 userpassword 변수에 할당한다. 비밀번호에 슬래시(/)가 있다면 한글 가로 치환한다.

|  |
| --- |
| NOTE |
| 비밀번호 재설정 메일에서 버튼을 누르면, 다음과 같은 형태의 웹 사이트 경로가 호출된다.  http://localhost:3000/PwChangeForm/이메일/토큰  토큰은 암호화된 비밀번호로 만든다. 만약 비밀번호 문자열에 슬래시(/)가 있을 경우, 웹 사이트 경로를 구분하는 슬래시(/)로 인식돼 인증되지 않는다. 그래서 암호화 결과 나올 수 없는 텍스트인 한글 가로 치환하고 이후에 토큰을 비교할 때 다시 슬래시(/)로 치환해 사용한다. |

30: 비밀번호가 정상적으로 조회됐다면, 이메일 발송 함수 sendEmail를 호출한다. 파라미터로 메일 주소, 메일 제목, 암호화된 비밀번호를 전달한다.

165. 비밀번호 재설정 이메일 발송 Api 호출하기

**• 학습 내용:** 이메일 발송에 필요한 파라미터를 세팅해 api를 호출한다.

**• 힌트 내용:** 수신자 이메일 주소, 메일 제목, 암호화된 비밀번호를 파라미터로 전달한다.

아이디(이메일)과 회원명으로 조회된 암호화된 비밀번호와 수신자 이메일 주소, 메일 제목을 파라미터에 담아 api를 호출한다.

components 폴더의 LoginForm.js 파일을 확인한다.

<소스> LoginForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18 | .. 코드 생략..  sendEmail = (email, subject, password, e) => {  axios.post('/api/mail', {  is\_Email : email,  is\_Subject : subject,  is\_Password: password  })  .then( response => {  if(response.data == "succ"){  this.sweetalert('입력하신 이메일로 비밀번호 \n'  + '재설정 메일 보내드렸습니다.', '', 'info', '닫기')  }  })  .catch( error => {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기');  });  }  .. 코드 생략.. |

3~7: 이메일 발송 함수(sendEmail)가 실행되면, 이메일 발송 api(/api/mail)를 호출한다. 이때 수신자 이메일 주소(is\_Email), 메일 제목(is\_Subject), 암호화된 비밀번호(is\_Password)를 파라미터로 전달한다.

9~12: 이메일 발송이 정상적으로 완료되면, 발송이 완료됐다는 알림 메시지를 표시한다.

비밀번호 재설정 페이지에서 정상적으로 아이디(이메일)와 회원명을 입력했는데도 다음과 같이 에러 메시지가 표시될 수 있다.

|  |
| --- |
|  |

**<그림 230>**

cmd 창에 출력된 node 서버의 로그를 보면, 다음과 같이 google 계정 관련 에러를 확인할 수 있다.

|  |
| --- |
|  |

**<그림 231>**

166. GMAIL 계정 접근 권한 허용하기

(저자 질문: 권한을 부여한다. 권한을 허용한다. 다른 의미로 사용한 것인지요?)

**• 학습 내용:** 구글 계정에 대한 액세스 권한을 허용한다.

**• 힌트 내용:** 웹 사이트에서 구글 계정에 접근할 수 있도록 권한을 허용한다.

웹 사이트에서 구글 계정의 자원을 사용해 이메일을 발송한다. 구글에서는 기본 설정으로, 보안이 증명되지 않은 웹 사이트에서 구글 자원에 접근하는 것을 제한한다. 접근을 허용하기 위해 사용할 계정으로 로그인을 하고 다음 화면에서 권한을 허용한다.

보안 수준이 낮은 앱의 액세스(https://myaccount.google.com/lesssecureapps?pli=1) 링크로 접속해 다음과 같이 보안 수준이 낮은 앱 허용을 사용으로 변경한다.

|  |
| --- |
|  |

**<그림 232>**

내 Google 계정에 대한 액세스 허용(https://accounts.google.com/b/0/DisplayUnlockCaptcha) 링크로 접속한 후 [계속] 버튼을 눌러 권한을 허용한다.

|  |
| --- |
|  |

**<그림 233>**

다시 비밀번호 재설정 페이지에서 아이디(이메일)과 회원명을 작성한다. [확인] 버튼을 누르면, 다음과 같이 메일 발송이 완료됐다는 메시지가 표시된다.

|  |
| --- |
| **<그림 234>** |

167. 수신된 이메일과 버튼 링크 확인하기

**• 학습 내용:** 이메일 템플릿과 동일한 화면의 메일이 수신된 것을 확인한다.

**• 힌트 내용:** 수신된 메일에서 버튼 링크가 정상적으로 할당됐는지 확인한다.

전송한 html 이메일 템플릿과 동일한 화면을 수신된 메일에서 확인할 수 있다. 페이지에 포함된 버튼에는 웹 사이트로 다시 돌아올 수 있는 링크가 할당돼 있다.

수신된 메일을 확인하면, 다음과 같이 메일 제목, 보낸 사람, 받는 사람, 이메일 템플릿, 버튼이 정상적으로 표시되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 235>**

개발자 도구에서 [비밀번호 변경하기] 버튼의 링크를 확인해보면, 다음과 같이 아이디(이메일)과 토큰이 정상적으로 할당돼 있는 것을 확인할 수 있다.

http://localhost:3000/PwChangeForm/react200@naver.com/$2b$10$hF58DTwPRK2kX5

|  |
| --- |
|  |

**<그림 236>**

168. 이메일 토큰으로 사용자 인증하기

**• 학습 내용:** 회원 인증 node api를 호출해 이메일 토큰으로 사용자 인증을 한다.

**• 힌트 내용:** 인증이 된 경우, 비밀번호 변경 페이지 접근 권한을 부여한다.

새로운 비밀번호로 변경할 수 있는 페이지를 만들고 인증된 사용자만 접근을 허용한다.

components 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9 10  11  12  13 14 15 16 17 | .. 코드 생략..  import PwChangeForm from './PwChangeForm';  class App extends Component {  render () {  return (  <div className="App">  <HeaderAdmin/>  .. 코드 생략..  <Route path='/PwChangeForm/:email/:token' component={PwChangeForm} />  <Footer/>  </div>  );  }  }  .. 코드 생략.. |

11: 비밀번호 수정 페이지는 /PwChangeForm 경로로 호출하고 line 2에 선언된 PwChangeForm.js로 라우팅한다. 경로의 :email과 :token에는 각각 수신된 이메일의 버튼 링크에 있던 이메일과 토큰이 전달된다.

components 폴더의 PwChangeForm.js 파일을 확인한다.

<소스> PwChangeForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31 | .. 코드 생략..  class PwChangeForm extends Component {  constructor (props) {  super(props);  this.state = {  email: props.match.params.email,  token: props.match.params.token,  }  }  componentDidMount() {  let token = this.state.token.replace(/가/gi, "/");  axios.post('/api/LoginForm?type=emailtoken', {  is\_Email : this.state.email,  is\_Token : token,  })  .then( response => {  if(response.data.json[0].username == undefined){  window.location.replace('about:blank')  }  })  .catch( error => {  this.sweetalert('유효한 접속이 아닙니다.', error, 'error', '닫기')  setTimeout(function() {  window.location.replace('about:blank')  }.bind(this),1000  );  });  }  .. 코드 생략.. |

6~8: App.js에서 PwChangeForm 컴포넌트로 라우팅된, 경로(/PwChangeForm/:email/:token)에서 email과 token 변수를 전달받는다. 라우팅 경로에 포함된 변수는 props.match.param로 접근해 값을 가져올 수 있다. email과 token 변수를 같은 이름의 state 변수에 할당한다.

13: 이메일 템플릿에 삽입해 발송했던 token 값은 암호화된 비밀번호를 앞에서 20자 자른 값이다. 20자 안에 슬래시(/)가 있을 경우, 새로운 경로의 시작으로 인식되기 대문에 한글 가로 치환했다. 이때 치환했던 한글 가를 다시 슬래시(/)로 치환한다.

14~17: 이메일(is\_Email)과 토큰(is\_Token)으로 회원 인증을 하는 node api(/api/LoginForm?type=emailtoken)를 호출한다.

18~22: api 호출 결과 회원 정보가 조회되지 않으면, username 값이 undefined가 된다. 이때 비밀번호 수정하기 페이지에 접근을 막아야 하기 때문에 빈 화면(about:blank)으로 강제 이동시킨다.

24~28: api 호출 중 에러가 발생해도 동일하게 빈 화면(about:blank)으로 강제 이동시킨다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20  21 | .. 코드 생략..  }else if(type == "emailtoken"){  // 이메일 인증 후 token으로 사용자 인증  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'select';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'selectEmailTokenCheck';  router.use('/', dbconnect\_Module);  next('route')    } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

10~12: type 파라미터의 값이 emailtoken일 경우, 토큰 사용자 인증 mapper 정보를 req.body에 할당한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16 | .. 코드 생략..  <select id="selectEmailTokenCheck">  SELECT  username  , userorg  , useremail  , userpassword  , usermajor  , userphone  , userflag  FROM  react.react\_user  WHERE useremail = #{is\_Email}  AND userpassword like CONCAT('%', #{is\_Token}, '%')  </select>.. 코드 생략.. |

14: react\_user 테이블에 파라미터로 전달받은 is\_Email 변수와 useremail 컬럼이 일치하는 데이터를 조회한다.

15: 토큰은 userpassword 컬럼의 값을 앞에서 20자를 추출해 만든 변수다. Like 구문을 사용해 userpassword 컬럼 값에 is\_Token값이 포함되는지 확인한다. Like 구문은 비교할 문자열에 %를 붙여 사용하는데, CONCAT 함수를 사용해 %와 비교할 문자열 변수를 연결한다.

|  |
| --- |
| NOTE |
| CONCAT 함수는 여러 문자열을 하나로 합쳐주는 역할을 한다. 예를 들어, SELECT CONCAT('react', '2', '00') FROM DUAL 쿼리를 실행하면, react200이 조회된다. |

169. 비밀번호 수정 Api 만들기 - 라우터 분기, 쿼리 추가

**• 학습 내용:** 인증된 회원이 비밀번호를 수정할 수 있는 api를 생성한다.

**• 힌트 내용:** 회원 가입과 동일하게 bcrypt를 사용해 비밀번호를 암호화한다.

비밀번호를 잊은 사용자가 이메일 인증을 완료하면, 새로운 비밀번호를 설정할 수 있다. 아이디(이메일)로 사용자 데이터를 조회해 비밀번호 컬럼을 변경한다.

routes 폴더의 UsersRout.js 파일을 확인한다.

<소스> UsersRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35 | .. 코드 생략..  const bcrypt = require('bcrypt');  const saltRounds = 10;  .. 코드 생략..  }else if(type == "pwdmodify"){  //비밀번호 재설정  try {  // Mysql Api 모듈(CRUD)  var dbconnect\_Module = require('./dbconnect\_Module');  //Mysql 쿼리 호출 정보 입력  req.body.mapper = 'UserMapper';//mybatis xml 파일명  req.body.crud = 'update';//select, insert, update, delete 중에 입력  req.body.mapper\_id = 'updatePwdUser';    var myPlaintextPassword = req.body.is\_Password;  if(myPlaintextPassword != '' && myPlaintextPassword != undefined ){  bcrypt.genSalt(saltRounds, function(err, salt) {  bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {  req.body.is\_Password = hash;  router.use('/', dbconnect\_Module);  next('route')  });  });  }else{  router.use('/', dbconnect\_Module);  next('route')  }    } catch (error) {  console.log("Module > dbconnect error : "+ error);  }  }  .. 코드 생략.. |

2~3: 회원 가입 api와 동일하게 bcrypt 패키지를 require해 사용할 수 있도록 한다. bcrypt에서 지원하는 salt의 크기를 ‘10’으로 설정한다.

13~15: type 파라미터의 값이 pwdmodify일 경우, 비밀번호를 업데이트하는 mapper 정보를 req.body에 할당한다.

17~19: req.body.is\_Password는 암호화되지 않은 비밀번호이다. 값이 있다면 brypt 암호화를 실행한다.

19: genSalt 함수로 line 3에서 지정한 크기의 salt를 생성한다.

20: hash 함수로 salt 값을 사용해 암호화되지 않은 비밀번호(myPlaintextPassword)를 암호화한다. 암호화된 비밀번호가 콜백 함수에서 hash 변수로 반환된다.

21: 암호화된 비밀번호(hash)를 다시 req.body.is\_Password 변수에 할당해 다음 라우터(dbconnect\_Module)로 전달한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9 | .. 코드 생략..  <update id="updatePwdUser">  UPDATE react.react\_user  SET  userpassword = #{is\_Password}  WHERE useremail = #{is\_Useremail}  </update>  .. 코드 생략.. |

3~6: 파라미터로 전달받은 아이디(is\_Useremail)로 회원 정보 테이블(react\_user)을 조회해 userpassword 컬럼을 새로 생성한 암호화된 비밀번호로 수정한다.

170. 비밀번호 수정 Api 호출하기

**• 학습 내용:** 새로운 비밀번호를 입력받아 비밀번호 수정 api를 호출한다.

**• 힌트 내용:** 회원 가입과 동일하게 입력받은 비밀번호의 유효성을 검사한다.

비밀번호 수정 api를 호출하는 방식은 전달하는 파라미터 수만 줄었을 뿐, 회원 가입 api를 호출하는 방식과 동일하다. update 쿼리에 필요한 아이디(이메일)과 새로운 비밀번호를 파라미터로 전달한다.

components 폴더의 PwChangeForm.js 파일을 확인한다.

<소스> PwChangeForm.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17  18  19  20 21 22  23 24  25  26  27  28  29 30 31  32 33  34  35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50  51 52 53  54  55  56 57  58  59  60  61 62 63  64  65 66  67 | .. 코드 생략..  submitClick = async (e) => {  this.pwd\_val\_checker = $('#pwd\_val').val();  this.pwd\_cnf\_val\_checker = $('#pwd\_cnf\_val').val();  this.fnValidate = (e) => {  .. 코드 생략..  }  if(this.fnValidate()){  var jsonstr = $("form[name='frm']").serialize();  jsonstr = decodeURIComponent(jsonstr);  var Json\_form = JSON.stringify(jsonstr).replace(/\"/gi,'')  Json\_form = "{\"" +Json\_form.replace(/\&/g,'\",\"').replace(/=/gi,'\":"')+"\"}";    try {  const response = await fetch('/api/register?type=pwdmodify', {  method: 'POST',  headers: {  'Content-Type': 'application/json',  },  body: Json\_form,  });  const body = await response.text();  if(body == "succ"){  this.sweetalertSucc('비밀번호 수정이 완료됐습니다.', false)  setTimeout(function() {  this.props.history.push('/');  }.bind(this),1500  );  }else{  this.sweetalert('작업 중 에러가 발생했습니다.', '', 'error', '닫기')  }  } catch (error) {  this.sweetalert('작업 중 에러가 발생했습니다.', error, 'error', '닫기')  }  }  };  .. 코드 생략..  <form method="post" name="frm" action="">  <input type="hidden" id="is\_Useremail" name="is\_Useremail" value={this.state.email}/>  <div className="log\_box">  <div className="in\_ty1">  <span className="ic\_2">  <img src={require("../img/main/m\_log\_i2.png")} alt="" />  </span>  <input type="password" id="pwd\_val"  name="is\_Password" placeholder="새 비밀번호" />  </div>  <div className="in\_ty1">  <span className="ic\_2">  <img src={require("../img/main/m\_log\_i2.png")} alt="" />  </span>  <input type="password" id="pwd\_cnf\_val"  name="is\_Password" placeholder="새 비밀번호 확인" />  </div>  <div className="btn\_confirm btn\_confirm\_m">  <Link to={'/'}>  <div className="bt\_ty bt\_ty\_m bt\_ty1 cancel\_ty1">취소</div>  </Link>  <a href="#n" className="bt\_ty bt\_ty\_m bt\_ty2 submit\_ty1"  onClick={(e) => this.submitClick(e)}>재설정</a>  </div>  </div>  </form>  .. 코드 생략.. |

62~63: 새로운 비밀번호를 입력한 후 [재설정] 버튼을 누르면 line 2의 submitClick 함수를 실행한다.

3~4: 비밀번호와 비밀번호 확인 입력 값을 각각 변수에 할당한다.

6~9: 회원 가입과 동일하게 비밀번호 유효성 검사를 한다. 입력한 비밀번호가 8~16자이면서 영문, 숫자, 특수 문자를 모두 포함했는지 검사한다.

11~15: 유효성 검사를 통과했다면 <form> 태그 안에 위치한 <input> 태그의 데이터를 json 형태로 변환한다. 49, 56에서 두 <input> 태그의 name 값이 password으로 동일한데, 동일한 값이 입력되기 때문에 문제되지 않는다.

18~24: 비밀번호 수정 api(/api/register?type=pwdmodify)를 호출한다. 파라미터로 <form> 태그 안에 위치한 아이디(is\_Useremail)와 입력한 비밀번호(is\_Password)가 전달된다.

25~32: 비밀번호 수정 api가 정상 동작했다면, response로 succ 문자열이 반환된다. 이때 비밀번호 수정 완료 메시지를 표시하고 로그인 페이지(/)로 이동한다.

이메일에서 [비밀번호 변경하기] 버튼을 누르면, 비밀번호 수정 페이지(/PwChangeForm)로 이동한다. 새로운 비밀번호 입력 후 [재설정] 버튼을 누르면, 다음과 같이 비밀번호 수정 완료 메시지가 표시된다. 로그인 페이지로 이동해 변경된 비밀번호로 로그인되는 것을 확인한다.

|  |
| --- |
|  |

**<그림 237>**

171. 스케줄러 만들기 1 - 배치 테이블 생성하기

**• 학습 내용:** 스케줄러를 사용해 데이터를 insert할 db 테이블을 생성한다.

**• 힌트 내용:** 스케줄러가 설정한 시간 단위에 쿼리가 실행된다.

스케줄러란, 정해진 시간에 어떤 작업을 수행하는 시스템이다. 배치 테이블은 웹 사이트에서 데이터를 조회하지 않고 단순히 스케줄러 테스트를 목적으로 사용한다.

|  |
| --- |
| NOTE |
| 배치(batch)란, 일괄적으로 작업을 처리하는 방식을 의미한다. 요청이 있을 때마다 작업하는 것과 반대로 정해진 시간에 일괄적으로 많은 작업을 처리한다. |

workbench에서 다음 쿼리를 실행한다.

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10 | use react;  CREATE TABLE `react\_batch\_log` (  `batch\_cd` varchar(100) COMMENT '배치 코드',  `batch\_nm` varchar(100) DEFAULT NULL COMMENT '배치명',  `batch\_log` varchar(100) DEFAULT NULL COMMENT '배치내용',  `reg\_date` varchar(100) DEFAULT NULL COMMENT '등록날짜',  PRIMARY KEY (`batch\_cd`)  );  ALTER TABLE react.react\_batch\_log convert to charset utf8; |

1: 웹 사이트에서 사용하는 스키마인 react를 사용하기 위해 use 명령어를 실행한다.

2: create table 쿼리를 실행해 배치 테이블(react\_batch\_log)을 생성한다.

8: 테이블에서 유일한 값을 갖는 batch\_cd 컬럼을 기본 키로 지정한다. batch\_cd 컬럼에는 현재 날짜 정보가 들어가 중복이 발생할 수 없다.

10: 테이블에 한글 데이터를 삽입할 수 있도록 charset을 utf8로 변경한다.

workbench에서 세미콜론(;)을 기준으로 3개의 쿼리를 순서대로, 다음과 같이 실행한다.

|  |
| --- |
|  |

**<그림 238>**

172. 스케줄러 만들기 2 - node-cron 패키지 사용

**• 학습 내용:** node-cron 패키지로 스케줄러를 구현할 수 있다.

**• 힌트 내용:** cron 표현식으로 쿼리 실행 시간을 설정할 수 있다.

cron은 고정된 날짜, 시간 주기로 작업을 실행할 수 있는 스케줄러다. Cron 표현식은 왼쪽부터 분, 시 일, 달, 요일을 나타낸다.

|  |
| --- |
| NOTE |
| Cron 표현식의 예시는 다음과 같다. \*은 모든 값을 ?은 특정 값이 없음을 뜻한다.  \* \* \* \* \* : 1분마다 실행  5 \* \* \* \* : 매시간 5분마다 실행(1시 5분, 2시 5분…)  10,44 14 ? 3 WED : 3월 동안 매주 수요일 오후 2:10분과 2:44분에 실행 |

cmd 창을 열어 node 경로(C:\react200)로 이동한다. [npm install --save node-cron]를 입력하면 다음과 같이 npm이 node-cron을 설치한다.

|  |
| --- |
|  |

**<그림 239>**

node 경로(C:\react200)의 server.js 파일을 확인한다.

<소스> server.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | var express = require('express');  .. 코드 생략..  var MailRout = require("./routes/MailRout");  require("./routes/BatchRout");  var app = express();  .. 코드 생략..  const port = process.env.PORT || 5000;  app.listen(port, () => console.log(`Listening on port ${port}`)); |

5: 스케줄러는 url을 호출받아 실행되는 것이 아니기 때문에 스케줄러를 정의할 파일 경로(/routes/BatchRout)만 require한다.

routes 폴더의 BatchRout.js 파일을 확인한다.

<소스> BatchRout.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34 35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50 | var express = require('express');  var router = express.Router();  var cron = require('node-cron');  // cron.schedule('5 \* \* \* \*', () => {  // cron.schedule('5 0 \* \* \*', () => {  cron.schedule('\* \* \* \* \*', () => {  var mapper = 'BatchMapper';//mybatis xml 파일명  var crud = 'insert';//select, insert, update, delete 중에 입력  var mapper\_id = 'insertBatchlog';  var param = { is\_Batchnm: '테스트 배치', is\_Batchlog: '테스트 배치가 정상적으로 실행됐습니다.' };  console.log('####### 배치 실행 / 테스트 배치 #######');  const mysql = require('mysql');  const mybatisMapper = require('mybatis-mapper');  const connection = mysql.createConnection({  host: "react200.#########.ap-northeast-2.rds.amazonaws.com",  port: "3306",  database: 'react',  user: "admin",  password: "react200RDS"  });  mybatisMapper.createMapper(['./models/'+mapper+'.xml']);  var time1 = new Date();  console.log('## '+time1+ ' ##');  console.log("\n Called Mapper Name = "+mapper);  var format = { language: 'sql', indent: ' ' };  var query = mybatisMapper.getStatement(mapper, mapper\_id, param, format);  console.log("\n========= Node Mybatis Query Log Start =========");  console.log("\* mapper namespce : "+mapper+"."+mapper\_id+" \*\n");  console.log(query+"\n");  connection.connect();  connection.query(query, function (error, results, fields) {  if (error) {  console.log("db error\*\*\*\*\*\*\*\*\*\*\*\*\* : "+error);  }  var time2 = new Date();  console.log('## '+time2+ ' ##');  console.log('## RESULT DATA LIST ## : \n', results);  console.log("========= Node Mybatis Query Log End =========\n");  });  connection.end();  });  module.exports = router; |

3: node-cron 패키지를 require해 사용할 수 있도록 한다.

5: 매시 5분마다 실행하는 cron 표현식이다. 예를 들면, 1시 5분, 2시 5분, 3시 5분에 실행된다.

6: 매일, 0시 5분마다 실행하는 cron 표현식이다. 예를 들면, 오늘 0시 5분, 내일 0시 5분, 모레 0시 5분에 실행된다.

8~10: 배치 테이블에 insert 쿼리를 실행하기 위해 쿼리가 정의된 mapper 정보를 각각 변수에 할당한다.

12: 배치 insert 쿼리에서 사용할 파라미터에 값을 할당한다.

18~24: 배치에서는 connection pool을 사용하지 않고 기본 mysql 모듈만 사용해 db connection을 만든다.

38~46: insert 쿼리가 정상적으로 실행됐다면, 배치 테이블에 데이터가 삽입된다. 그래서 쿼리를 실행하고 결과에 대해서 response를 보내지 않아도 된다.

173. 스케줄러 만들기 3 - 특정 시간에 insert 쿼리 실행하기

**• 학습 내용:** 배치 mapper 파일에 insert 쿼리를 추가한다.

**• 힌트 내용:** 전달받은 파라미터들을 데이터 형태에 맞게 삽입한다.

회원 가입 페이지에서 나눠 전달한 휴대전화 번호와 이메일 주소를 mysql CONCAT 함수를 사용해 가공해 삽입한다.

models 폴더의 UserMapper.xml 파일을 확인한다.

<소스> UserMapper.xml

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13  14 15  16  17 | .. 코드 생략..  <insert id="insertBatchlog">  INSERT INTO react.react\_batch\_log  (  batch\_cd  , batch\_nm  , batch\_log  , reg\_date  )  VALUES (  CONCAT("BL",DATE\_FORMAT(now(), '%Y%m%d%H%i%s'))  , #{is\_Batchnm}  , #{is\_Batchlog}  , DATE\_FORMAT(now(), '%Y%m%d%H%i%s')  )  </insert>.. 코드 생략.. |

12: 기본 키 batch\_cd 컬럼의 값으로, BL 문자열에 현재 시간을 붙여 삽입한다.

13: BatchRout.js에서 전달한 배치명을 삽입한다.

14: BatchRout.js에서 전달한 배치 내용을 삽입한다.

1분마다 다음과 같이 insert 쿼리가 실행되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 240>**

workbench에서 적재된 데이터를 확인해보면, 1분 간격으로 데이터가 삽입된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 241>**

174. 웹 서버 구축하기 1 - AWS EC2 인스턴스 생성

**• 학습 내용:** AWS의 EC2 인스턴스를 프리티어(t2.micro) 무료 버전으로 생성한다.

**• 힌트 내용:** EC2 인스턴스는 react, node 코드를 배포해 웹 서버로 사용한다.

AWS가입 후 EC2 인스턴스를 생성한다. 기존 RDS 인스턴스를 추가해 사용 중인 계정에서 EC2 인스턴스를 생성하는 것이 관리하게 편리하다. AWS 웹 사이트(<https://aws.amazon.com/ko/>)에서 가입한 계정으로 로그인을 하고 [내 계정] - [AWS Management Console]을 클릭한다.

1. 왼쪽 상단의 [서비스] - [컴퓨팅] - [EC2]를 클릭한다. 다음과 같이 EC2 Dashboard 페이지에서 [인스턴스 시작] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 242>**

2. 1단계 [AMI 선택]에서 Ubuntu Server 18.04 LTS (HVM)를 선택한다. 로컬에서는 윈도우 운영체제에서 react와 node 서버를 구동했다. AWS에서는 운영체제가 ubuntu로 바뀌었을 뿐, 로컬과 동일한 방식으로 서버를 구동할 수 있다.

|  |
| --- |
|  |

**<그림 243>**

3. 2단계 [인스턴스 유형 선택]에서 프리티어 무료 버전인(t2.micro)를 선택한 후 [다음:인스턴스 세부 정보 구성] 버튼을 누른다. 같은 계정에 RDS와 EC2 서버를 동시에 프리티어(t2.micro)로 사용하더라도 1년간 무료로 사용할 수 있다.

|  |
| --- |
|  |

**<그림 244>**

4. 3단계 [인스턴스 구성], 4단계 [스토리지 추가], 5단계 [태그 추가]에서는 변경사항 없이 [다음:인스턴스 세부 정보 구성] 버튼을 누른다.

175. 웹 서버 구축하기 2 - 포트 설정, 키 페어 다운로드

**• 학습 내용:** AWS의 EC2 인스턴스에 접속할 수 있는 포트를 설정하고 키 페어를 다운로드한다.

**• 힌트 내용:** SFTP와 SSH를 통해 EC2 인스턴스에 접속하기 위해 키 페어를 사용한다.

보안 그룹은 인스턴스에 접근하는 트래픽을 관리하는 규칙이다. 설정한 포트를 통해서만 EC2 서버에 접속할 수 있도록 한다. 생성된 EC2 서버에 파일을 전송하려면 SFTP를 사용하는데, 로컬에서 폴더 구조와 같다. 로컬에서는 cmd 창을 열어 npm 명령어를 실행했는데, EC2 서버에서는 SSH를 cmd 창과 같이 사용한다. SFTP와 SSH로 EC2 서버에 접근할 수 있도록 키 페어를 통해 인증한다.

1. 6단계 [보안 그룹 구성]에서 [규칙 추가] 버튼을 눌러 다음과 같이 3000포트를 추가한다.

|  |
| --- |
|  |

**<그림 245>**

2. 7단계 [인스턴스 시작 검토]에서 오른쪽 하단의 [시작하기] 버튼을 누르면, 다음과 같이 키 페어 생성 창이 표시된다. 이때 [새 키 페어 생성]을 선택하고 키 페어 이름을 정해 다운로드한다. 키 페어는 서버 자원에 직접 접근할 수 있도록 인증하는 중요한 키다. 외부에 유출되거나 잃어버리지 않도록 관리한다. 키 페어 다운로드를 완료하고 [인스턴스 시작] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 246>**

3. 생성된 인스턴스를 확인하기 위해 EC2 대시보드에서 [실행 중인 인스턴스]를 클릭한다.

|  |
| --- |
| **<그림 247>** |

4. 이동한 페이지에서 다음과 같이 인스턴스가 생성된 것을 확인할 수 있다. 인스턴스 상태가 running이 되면 사용할 수 있다.

|  |
| --- |
|  |

**<그림 248>**

176. 웹 서버 구축하기 3 - 탄력적 IP 설정하기

**• 학습 내용:** AWS의 EC2 서버의 IP를 고정적으로 사용하기 위해 탄력적 IP를 설정한다.

**• 힌트 내용:** 탄력적 IP를 설정하지 않고 EC2 서버를 재부팅하면 IP가 바뀐다.

EC2 인스턴스를 생성하면, 외부에 접근할 수 있는 퍼블릭 IP가 할당된다. 서버를 종료한 후 시작하거나 재부팅하면 서버에 할당됐던 퍼블릭 IP가 변한다. 서버 IP가 계속 바뀌면 여러 가지 번거로운 상황이 발생한다. 서버 자원에 접근하기 위해 SFTP, SSH를 사용할 경우와 도메인이 연결된 경우 매번 IP주소를 바꿔줘야 한다. 탄력적 IP를 사용하면 서버가 재시작되더라도 고정된 IP 주소를 사용할 수 있다.

1. EC2 인스턴스를 생성하면 다음과 같이 [IPv4 퍼블릭 IP] 컬럼에 외부에서 접속할 수 있는 IP가 할당된다. [탄력적 IP] 컬럼의 값은 비어 있다는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 249>**

2. EC2 대시보드 왼쪽 메뉴에서 [탄력적 IP]를 클릭하면, 다음과 같이 등록된 탄력적 IP 주소가 없는 것을 확인할 수 있다. [탄력적 IP 주소 할당] 버튼을 눌러 상세 페이지로 이동한다.

|  |
| --- |
|  |

**<그림 250>**

3. 탄력적 IP 주소 할당 상세 페이지로 이동하면, 다음과 같이 [Amazon의 IPv4 주소 풀]에 기본으로 선택돼 있다. 이 상태에서 [할당] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 251>**

4. 다음과 같이 추가된 탄력적 IP의 체크 박스를 선택하고 [작업] - [탄력적 IP 주소 연결] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 252>**

5. [탄력적 IP 주소 연결] 페이지로 이동하면, 다음과 같이 [인스턴스]와 [프라이빗 IP 주소] 영역을 클릭한다. 탄력적 IP를 연결할 인스턴스의 ID와 프라이빗 IP를 선택한 후 [연결] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 253>**

6. 정상적으로 탄력적 IP가 연결됐다면, 다음과 같이 [탄력적 IP] 컬럼의 값이 할당된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 254>**

177. MobaXterm - SFTP로 EC2 서버 접속하기

**• 학습 내용:** SFTP로 EC2 서버에 접속해 폴더와 파일 구조를 확인한다.

**• 힌트 내용:** SFTP 사용을 위해 MobaXterm 프로그램을 설치한다.

FTP(파일 전송 프로토콜)은 서버와 클라이언트 사이에서 파일 직접 전송을 가능하게 하는 프로토콜이다. SFTP에서 S는 secure로 FTP보다 보안이 강화된 전송 방식을 의미한다. SFTP를 사용하면, 로컬에서 개발한 코드를 간편하게 EC2 서버로 전송할 수 있다. SFTP를 사용하기 위한 툴로 MobaXterm을 사용한다.

다음 링크에서 MobaXterm을 다운로드하고 설치한다.

https://mobaxterm.mobatek.net/download.html

1. MobaXterm을 실행하고 왼쪽 상단의 [Session] 버튼을 누르면, 다음과 같은 팝업 창이 표시된다. 팝업 창 상단의 기능들 중 [SFTP]를 선택한다. [Remote host]에는 탄력적 IP를 입력하고 [Username]은 EC2 서버의 기본값인 ubuntu를 입력한다. 하단의 [Advanced Sftp settings] 탭을 선택하고 [Use private key] 체크 박스를 클릭한다. 이때 [Use private key] 버튼 오른쪽 영역의 파일 선택 기능을 사용해 EC2 서버의 키 페어 파일을 업로드한다. [OK] 버튼을 눌러 SFTP로 EC2 서버에 접속한다.

|  |
| --- |
|  |

**<그림 255>**

2. SFTP 연결이 정상적으로 완료되면, 하단에 SSH 인증에 성공해 SFTP 연결을 시작한다는 메시지가 표시된다. 다음과 같이 EC2 서버의 기본 경로(/home/ubuntu)에 접근할 수 있다. MobaXterm 왼쪽 Sessions 영역을 보면 추가된 SFTP 세션을 확인할 수 있는데, 한 번만 추가해 놓으면 클릭만으로 간편하게 SFTP 세션을 다시 연결할 수 있다. 또 여러 개의 세션을 추가해 사용할 수 있다.

|  |
| --- |
|  |

**<그림 256>**

178. MobaXterm - SSH로 EC2 서버 접속하기

**• 학습 내용:** SSH로 EC2 서버에 접속해 ubuntu 터미널을 사용한다.

**• 힌트 내용:** SSH 사용을 위해 MobaXterm 프로그램을 사용한다.

SSH(Secure Shell)은 서로 다른 컴퓨터에 네트워크를 통해 접근하고 명령어를 실행할 수 있게 해주는 프로토콜이다. 암호화 기법을 사용하기 때문에 보안적으로 안전하게 사용할 수 있다. 로컬에서 cmd 창을 사용해 실행하던 npm 명령어를 EC2 서버에서는 SSH를 통해 실행할 수 있다. SSH를 사용하기 위한 툴로 MobaXterm을 사용한다.

1. MobaXterm을 실행한 후 왼쪽 상단의 [Session] 버튼을 누르면, 다음과 같은 팝업 창이 표시된다. 팝업 창 상단의 기능들 중 [SSH]를 선택한다. [Remote host]에는 탄력적 IP를 입력하고 [Username]은 EC2 서버의 기본값인 ubuntu를 입력한다. 하단의 [Advanced Sftp settings] 탭을 선택하고 [Use private key] 체크 박스를 클릭한다. 이때 [Use private key] 버튼 오른쪽 영역의 파일 선택 기능을 사용해 EC2 서버의 키 페어 파일을 업로드한다. [OK] 버튼을 눌러 SSH로 EC2 서버에 접속한다.

|  |
| --- |
|  |

**<그림 257>**

2. SSH 연결이 정상적으로 완료되면, 다음과 같이 ubuntu 터미널을 사용할 수 있다. MobaXterm 왼쪽 Sessions 영역을 보면 추가된 SSH 세션을 확인할 수 있다. SFTP와 마찬가지로 한 번만 추가해 놓으면, 클릭만으로 간편하게 SSH 세션을 다시 연결할 수 있다.

|  |
| --- |
|  |

**<그림 258>**

179. SFTP로 EC2 서버에 로컬 소스 옮기기

**• 학습 내용:** 배포할 로컬 소스를 SFTP를 사용해 EC2 서버 디렉터리로 옮긴다.

**• 힌트 내용:** MobaXterm 프로그램을 사용하면, 드래그로 간편하게 업로드할 수 있다.

로컬에서 개발한 소스를 EC2 서버에 배포하기 위해서는 먼저 로컬 소스를 EC2 서버 디렉터리로 옮겨야 한다. 로컬 소스를 tar 확장자로 압축하고 SFTP 창으로 드래그하면 파일을 간편하게 EC2 서버 경로로 업로드할 수 있다. 윈도우 운영체제에서는 파일을 압축할 때 zip 파일을 많이 사용하지만, ubuntu와 같은 리눅스 운영체제에서는 tar 파일을 사용한다.

1. 기존 개발 소스의 node 경로에서 node\_modules 폴더와 client 폴더를 제외하고 새로운 폴더로 복사한다. client 폴더를 새로 생성한다. node\_modules 폴더와 client/node\_modules 폴더에는 각각 node와 react에서 사용하는 패키지들이 설치돼 있기 때문에 파일을 옮기는 데 오랜 시간이 걸린다. 그래서 두 경로의 파일들은 제외하고 EC2 서버에 업로드한다. 필요한 패키지들은 package.json 파일에 작성돼 있기 때문에 npm install 명령어로 한 번에 설치할 수 있다.

|  |
| --- |
|  |

**<그림 259>**

2. 새로 생성한 client 폴더에 기존 개발 react 경로 소스에서 node\_modules 폴더를 제외하고 복사한다.

|  |
| --- |
|  |

**<그림 260>**

3. 복사한 폴더의 node 경로에서 모든 폴더와 파일을 선택하고 tar 파일로 압축한다.

|  |
| --- |
|  |

**<그림 261>**

4. MobaXterm 프로그램에서 SFTP로 EC2 서버에 접속한다. 다음과 같이 /home/ubuntu 경로에 tar 파일을 업로드할 폴더(react200)을 생성한다.

|  |
| --- |
|  |

**<그림 262>**

5. SFTP 창에서 생성한 폴더(react200)를 열고 로컬 소스를 압축한 tar 파일을 SFTP 폴더 영역으로 드래그해 놓는다. 다음과 같이 tar 파일이 SFTP 폴더 경로에 업로드된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 263>**

180. SSH로 EC2 서버에 소스 압축 풀기, nodejs 설치하기

**• 학습 내용:** 업로드한 tar 파일의 압축을 풀고 EC2 서버 ubuntu 운영체제에 nodejs를 설치한다.

**• 힌트 내용:** ubuntu 운영체제의 패키지 관리 도구인 apt를 사용해 nodejs와 npm을 설치한다.

tar 파일의 압축을 풀면, 로컬과 동일한 디렉터리 구조가 생성된다. EC2 인스턴스를 생성하면 아무것도 설치돼 있지 않기 때문에 nodejs를 설치해야 한다. 윈도우에서는 nodejs를 설치하면 npm이 함께 설치됐지만, ubuntu에서는 npm도 별도로 설치해야 한다.

1. SSH로 EC2 서버 터미널에 접속한다. root 계정으로 전환[sudo su –] 후 tar 파일이 있는 경로로 이동한다[cd /home/ubuntu/react200]. 그리고 업로드한 tar 파일의 압축을 해제한다[tar -xvf 179.tar].

|  |
| --- |
| NOTE |
| root 계정은 관리자 계정이다. 관리자 권한으로만 실행할 수 있는 명령어가 있기 때문에 터미널을 사용할 때는 root 계정으로 전환한 후에 사용하는 것이 좋다. |

|  |
| --- |
|  |

**<그림 264>**

2. 압축을 해제하고 SFTP로 tar 파일이 있던 폴더에 접속하면, 다음과 같이 로컬과 동일한 디렉터리 구조가 생성된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 265>**

3. ubuntu 운영체제에서는 어떤 프로그램을 설치할 때 apt라는 패키지 관리 도구를 사용한다. [sudo apt-get update] 명령어로 설치된 패키지들을 최신 버전으로 업데이트한다. [sudo apt-get install -y nodejs] 명령어로 nodejs를 설치한다.

|  |
| --- |
|  |

**<그림 266>**

4. [apt install npm] 명령어로 npm을 설치한다.

|  |
| --- |
|  |

**<그림 267>**

5. [node -v], [npm -v] 명령어로 설치된 nodejs와 npm의 버전을 확인한다.

|  |
| --- |
|  |

**<그림 268>**

181. react, node 서버에 필요한 패키지 설치하기

**• 학습 내용:** react, node 서버 경로에서 각각, package.json 파일에 작성된 패키지들을 설치한다.

**• 힌트 내용:** 패키지들이 설치되면, 로컬 경로와 동일하게 node\_modules 폴더가 생성된다.

로컬에 있는 파일을 EC2 경로에 옮길 때 실제 패키지들이 설치된 node\_modules 폴더를 제외했다. 패키지 파일들을 옮기는 것보다 다시 설치하는 것이 빠르기 때문에 react, node 경로에서 각각 필요한 패키지를 새로 설치한다.

1. SSH로 EC2 서버 터미널에 접속한다. root 계정으로 전환[sudo su –]한 후 node 경로로 이동한다[cd /home/ubuntu/react200]. 그리고 package.json에 기록돼 있는 패키지(node)들을 [npm install] 명령어로 설치한다. 설치 중 bcrypt에 관련된 설치 에러가 발생한다.

|  |
| --- |
|  |

**<그림 269>**

2. 다음과 같이 npm 웹 사이트의 bcrypt 페이지에는 호환되는 bcrypt 버전과 node 버전이 정리돼 있다. 예제에서 설치한 node 버전은 8.10.0인데, bcrypt 버전은 5.0.0으로 호환되지 않는다.

|  |
| --- |
| **<그림 270>** |

3. [npm install bcrypt@3.0.6] 명령어로 1.0.3보다 크고 4.0.0보다 작은 버전의 bcrypt를 설치한다.

|  |
| --- |
|  |

**<그림 271>**

4. [npm install nodemon] 명령어를 실행해 node 서버 구동에 필요한 nodemon 패키지를 설치한다.

|  |
| --- |
|  |

**<그림 272>**

5. [cd client]로 react 경로로 이동한다. [curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -], [echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list]를 실행해 yarn 공식 repository를 추가한다. [apt-get update]로 apt 버전을 최신화하고 [apt-get install yarn] 명령어로 yarn을 설치한다. [yarn install] 명령어로 package.json에 기록돼 있는 패키지(react)들을 설치한다.

|  |
| --- |
|  |

**<그림 273>**

182. react, node 서버 구동해 외부에서 접속하기

**• 학습 내용:** yarn dev 명령어로 react, node 서버를 구동하고 웹 브라우저에서 확인한다.

**• 힌트 내용:** EC2 서버의 탄력적 IP에 포트 3000을 붙여, 웹 사이트의 홈 경로(/)를 호출한다.

react와 node 경로에 필요한 패키지들이 설치됐다면, 로컬에서 사용하던 것과 동일하게 디렉터리 구조가 생성된다. SFPT와 SSH를 연결할 때 사용하던 탄력적 IP를 그대로 웹 사이트에 접속할 때도 사용한다.

1. [cd..]으로 상위 경로인 node 경로로 이동한다. 로컬에서 서버를 구동하던 명령어 [yarn dev]를 그대로 사용해 EC2 서버에서 react와 node 서버를 구동한다.

|  |
| --- |
|  |

**<그림 274>**

|  |
| --- |
| NOTE |
| 예제 파일에는 DB 서버 정보가 없어 그대로 업로드했을 경우, DB 서버 연결 에러가 발생한다. dbconnect\_Module.js와 BatchRout.js 소스에서 mysql 연결 정보(host, user, password)를 각자의 RDS 정보로 입력해야 한다. |

2. 웹 브라우저에서 EC2 서버의 탄력적 IP에 react 기본 포트인 3000을 붙여 호출한다. 그러면 다음과 같이 웹 사이트에 접속할 수 있다. 로컬(127.0.0.1:3000)에서 서버를 구동했을 때와 동일한 화면이 표시되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 275>**

183. EC2 서버 시간 한국으로 변경하기

**• 학습 내용:** date 명령어로 EC2 서버 시간을 확인하고 한국 시간으로 변경한다.

**• 힌트 내용:** 서버 시간은 SSH에서 쿼리 실행 시간 등을 로그로 출력할 때 표시된다.

EC2 서버를 생성하면, 서버 시간이 UTC(협정 세계 표준시)로 설정돼 있다. 웹 사이트에서 쿼리 실행 시간 등의 로그를 출력하거나 서버 시간 정보가 개발 로직에 필요할 수 있다. 기존의 UTC로 설정된 localtime 정보를 삭제하고 서울 시간으로 변경한다.

1. SSH로 접속해 [date] 명령어를 실행하면, 다음과 같이 UTC 시간 정보가 출력된다. [sudo rm /etc/localtime] 명령어로 기존 localtime 정보를 삭제하고 [sudo ln -s /usr/share/zoneinfo/Asia/Seoul /etc/localtime] 명령어로 localtime 정보를 서울로 변경한다.

|  |
| --- |
|  |

**<그림 276>**

2. node 경로에서 [yarn dev] 명령어로 서버를 구동하면, 다음과 같이 한국 시간으로 로그가 출력되는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 277>**

184. vi 에디터로 EC2 서버 소스 직접 수정하기

**• 학습 내용:** SSH 접속 후 vi에디터를 사용해 서버 파일을 직접 수정한다.

**• 힌트 내용:** 매분마다 실행되는 스케줄러의 실행 시간 정보를 매일 1번만 실행하도록 수정한다.

운영되고 있는 서버의 소스를 직접 수정하는 것은 위험할 수 있다. 실무에서는 로컬에서 개발한 소스를 git, svn 등을 사용해 버전 관리를 하고 서버에 배포한다. 이렇게 하면 로컬 소스와 서버 소스가 일치하기 때문에 서버에 문제가 발생했을 때 로컬에도 원인을 파악할 수 있을 확률이 크다. 또 서버에 배포할 코드는 로컬에서 충분히 테스트를 해보고 올리는 것이 안전하다. 하지만 불가피하게 서버의 소스를 직접 변경해야 할 경우가 발생하기도 한다. 이때 vi 에디터를 사용해 코드를 변경할 수 있다.

1. root 계정으로 전환[sudo su –]한 후 node 경로로의 routes 폴더로 이동한다[cd /home/ubuntu/react200/routes]. routes 폴더에 있는 BatchRout.js 파일을 vi 에디터로 열기 위해 [vi BatchRout.js] 명령어를 실행한다.

|  |
| --- |
|  |

**<그림 278>**

2. 다음과 같이 BatchRout.js 파일이 열린다. 이때 코드 수정을 위해 키보드에서 [i]를 누르면, 왼쪽 아래에 [-- INSERT --] 문구가 표시된다. 이 상태에서는 키보드 방향키로 마우스 커서를 움직여 코드를 수정할 수 있다. 매분마다 실행되도록 돼 있는 cron 패턴(\* \* \* \* \*) 코드로 커서를 이동한다. 키보드 [backspace]로 기존 코드를 삭제한 후 매일 00시 5분에 실행되는 패턴(5 0 \* \* \*)으로 코드를 수정한다.

|  |
| --- |
|  |

**<그림 279>**

3. 수정을 완료한 후 [Esc]를 누르면, 수정 모드가 해제돼 [-- INSERT --] 문구가 사라진다. 이때 [:wq]를 입력하고 [enter]를 누르면, 수정된 코드가 저장되고 터미널 화면으로 이동한다.

|  |
| --- |
|  |

**<그림 280>**

4. 다시 [vi BatchRout.js] 명령어로 파일을 열면, 수정된 코드가 적용된 것을 확인할 수 있다. Vi 에디터에 저장하지 않고 터미널로 나가려면, [:q]를 입력한 후 [Enter]를 누른다.

|  |
| --- |
|  |

**<그림 281>**

185. chown, chmod 명령어로 서버 접근 권한 부여하기

**• 학습 내용:** chown, chmod 명령어로 특정 계정에 원하는 권한을 부여할 수 있다.

**• 힌트 내용:** owner, group 권한별로 소유자를 정하고 일기, 쓰기, 실행 권한을 부여한다.

ubuntu 운영체제에서는 폴더와 파일마다 소유자와 권한이 정해져 있다. 소유자의 종류는 owner와 group이 있는데, [chown owner 소유자:group 소유자 폴더/ 파일명] 문법으로 지정할 수 있다. 폴더/ 파일의 권한은 [chmod 권한 숫자 폴더/ 파일명] 형태의 문법으로 부여할 수 있다. 이때 권한 숫자는 8진수 세 자리로 나타낸다. 권한은 [drwxrwx---]와 같이 표현되는데, 맨 앞의 d는 디렉터리(폴더)인 경우만 붙는다. 그다음 rwx가 3쌍이 나오는데 첫 번째 rwx는 own에 대한 것이고 두 번째는 group에 대한 권한이다. 세 번째(---)는 기타 사용자로 사용하지 않는다. rwx는 각각 읽기(read), 쓰기(write), 실행(execute)을 의미한다. rwx를 한 쌍으로 권한 숫자를 만드는데 r = 4, w = 2, x = 1, '-' = 0를 의미한다.

|  |
| --- |
| NOTE |
| 권한 숫자 예시는 다음과 같다.  -rwxrwxrwx : 777( 파일)  -rwxrw---- : 760( 파일)  drwxr-x--- : 757( 폴더)  d--xrwxr-- : 174( 폴더) |

1. 지난 예제에서 node 경로(/home/ubuntu/react200)에 tar 파일의 압축을 풀었다. SSH로 접속해 root 권한으로 압축을 풀었기 때문에 모든 폴더와 파일의 소유자(owner, group 모두)는 root이다. 그런데 SFTP는 ubuntu 권한으로 접속하기 때문에 node 경로에 있는 폴더와 파일을 수정할 수 없다. 이 상태에서 로컬에 있는 server.js 파일을 SFTP 창의 node 경로로 드래그하면, 다음과 같이 [Permession denied] 에러 메시지가 표시된다.

|  |
| --- |
|  |

**<그림 282>**

2. SSH 창에서 root 계정으로 전환[sudo su –]한 후 node 경로보다 한 단계 상위 경로로 이동한다[cd /home/ubuntu]. [chown -R root:ubuntu react200] 명령어로 react200 폴더의 owner 권한 소유자를 root 계정에 group 권한 소유자를 ubuntu 계정에 부여한다. [-R] 문법을 추가하면 지정한 폴더 하위에 존재하는 모든 폴더와 파일에 적용된다. [chmod -R 770 react200] 명령어를 실행해 react200 폴더와 하위 모든 폴더 및 파일의 권한을 rwxrwx---(770)으로 부여한다.

|  |
| --- |
|  |

**<그림 283>**

3. 권한 수정이 완료되고 SFTP에서 node 경로(/home/ubuntu/react200)를 확인해보면, 다음과 같이 권한과 소유자가 변경된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 284>**

186. nohup으로 서버 상시 구동하기, 포트로 서버 종료하기

**• 학습 내용:** nohup 명령어로 서버를 구동하면, SSH 창을 닫아도 서버가 꺼지지 않는다.

**• 힌트 내용:** 로컬에서 cmd 창을 닫으면 서버가 종료되는 것처럼 SSH 창도 동일하게 동작한다.

SSH 터미널에서 yarn dev 명령어로 react, node 서버를 구동하고 SSH 창을 닫으면 서버가 종료된다. ubuntu 운영체제에서는 nohup이라는 명령어를 사용해 터미널이 종료돼도 서버를 유지할 수 있다.

1. SSH 창에서 root 계정으로 전환[sudo su –]한 후 node 경로로 이동한다[cd /home/ubuntu/react200]. [nohup yarn dev &] 명령어로 서버를 실행한다.

|  |
| --- |
|  |

**<그림 285>**

2. MobaXterm 툴에서 SSH 창을 닫아도 다음과 같이 서버가 살아 있는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 286>**

3. nohup으로 구동한 서버를 종료할 때는 다음과 같이 포트 번호를 사용할 수 있다. root 계정으로 전환[sudo su –]한 후 [fuser -k 3000/tcp] 명령어를 실행하면 3000번 포트를 사용하는 서버(react 서버)를 종료할 수 있다. react 서버 종료 후 [yarn dev] 명령어로 서버를 재시작했을 때 5000포트가 이미 사용 중이라는 에러 메시지가 발생할 수 있다. 이런 경우 node 서버가 종료되지 않은 경우인데, [fuser -k 5000/tcp] 명령어로 5000번 포트를 사용하는 서버(node 서버)를 종료한다.

|  |
| --- |
|  |

**<그림 287>**

187. kakao 지도 API appkey 발급, 웹 사이트 도메인 등록하기

**• 학습 내용:** react에서 카카오 지도 API를 사용하기 위해서는 사전 작업이 필요하다.

**• 힌트 내용:** kakao map api 웹 사이트에서 appkey를 발급받고 사용할 웹 사이트의 도메인을 등록한다.

개발할 웹 사이트에서 지도 관련 기능을 추가할 때 지도 api를 호출해 사용하면 간편하게 구현할 수 있다. kakao 지도 api를 사용하기 위해서는 api 호출 시 발급받은 appkey를 파라미터로 전달해야 한다. 또 api를 호출할 웹 사이트의 도메인을 등록해야 한다. 로컬 서버의 경우 localhost를 등록하면 된다. kakao 지도에서 제공해주는 공식 가이드(<https://apis.map.kakao.com/web/guide/>)를 참고한다.

1. 카카오 개발자 웹 사이트 (<https://developers.kakao.com>)에 접속해 카카오 계정으로 로그인한다.

|  |
| --- |
|  |

**<그림 288>**

2. 상단의 [내 애플리케이션]을 클릭하면, 다음과 같은 화면으로 이동한다. [애플리케이션 추가하기] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 289>**

3. 앱명과 회사명을 입력하고 저장한다.

|  |
| --- |
|  |

**<그림 290>**

4. 추가된 애플리케이션이 표시되면, 애플리케이션명을 클릭한다.

|  |
| --- |
|  |

**<그림 291>**

5. 이동한 페이지에서 [앱 키] - [JavaScript 키]를 복사해둔다. react 서버에서 kakao map api를 호출할 때 [JavaScript 키]를 사용한다. 하단의 [플랫폼] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 292>**

6. 다음과 같이 플랫폼 등록 페이지 이동됐다면, 로컬에서 사용할 도메인(<http://localhost>)를 [Web] 플랫폼으로 등록한다. 등록된 도메인을 갖는 서버에서만 kakao map api를 호출할 수 있다.

|  |
| --- |
|  |

**<그림 293>**

188. react 서버 포트 80으로 설정하기

**• 학습 내용:** react 서버의 기본 포트 번호인 3000을 80으로 변경한다.

**• 힌트 내용:** 서버 포트를 80으로 설정하면, 포트 번호 없이 웹 사이트 주소를 호출할 수 있다.

create-react-app으로 프로젝트를 생성하면, 서버 기본 포트가 3000번이다. 포트 번호를 80번으로 변경하면, 웹 사이트 주소를 포트번호를 제외하고 호출할 수 있다. 기존에 로컬에서 서버를 구동할 때 localhost:3000으로 호출했다면, 80포트를 사용할 땐 localhost만 호출하면 된다.

1. 새로운 프로젝트 폴더를 생성하고 예제 파일을 옮겨 넣거나 create-react-app으로 프로젝트를 생성하고 예제 파일을 덮어쓰기한다.

|  |
| --- |
|  |

**<그림 294>**

2. react 경로(C:\react\_kakaomap)의 package.json 파일을 확인한다.

<소스> package.json

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8  9  10  11 | {  "name": "react\_kakaomap",  .. 코드 생략..  },  "scripts": {  "start": "set PORT=80 && react-scripts start",  "build": "react-scripts build",  "test": "react-scripts test",  "eject": "react-scripts eject"  },  .. 코드 생략.. |

6: cmd 창에 npm start 명령어를 실행하면, react-scripts가 시작되면서 react 서버가 켜진다. 이때 set PORT구분을 추가하면, 원하는 포트번호로 react 서버를 구동할 수 있다.

189. react에서 kakao 지도 생성하기

**• 학습 내용:** kakao map api를 호출하고 react 페이지에 지도를 표시한다.

**• 힌트 내용:** 표시되는 지도의 좌표, 확대 레벨, 크기를 조절할 수 있다.

kakao map api웹 사이트에서 appkey를 발급받고 웹 사이트 도메인을 등록했다면, 등록된 도메인에서 api를 호출할 수 있다.

public 폴더의 index.html 파일을 확인한다.

<소스> index.html

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13 | <!DOCTYPE html>  <html lang="en">  <head>  .. 코드 생략..  <title>React App</title>  <script type="text/javascript" src="//dapi.kakao.com/v2/maps/sdk.js?appkey=[키입력]">  </script>  </head>  <body>  .. 코드 생략..  </body>  </html> |

6~7: <script> 태그의 src 속성에 kakao map api url을 할당한다. 파라미터 appkey에는 kakao map api웹 사이트에서 발급 받은 [JavaScript 키] 값을 입력한다.

src 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19 | /\*global kakao\*/  import React, { Component } from 'react';  class App extends Component {  componentDidMount() {  var mapContainer = document.getElementById('map'),  mapOption = {  center: new kakao.maps.LatLng(36.2683, 127.6358),  level: 13  };  var map = new kakao.maps.Map(mapContainer, mapOption);  }  render() {  return (  <div id="map" style={{"width":"100%","height":"700px"}}></div>  )  }  }  export default App; |

15: 지도가 표시될 id가 map인 <div> 태그를 추가한다. style 속성으로 지도의 높이와 너비를 지정한다.

4: line 13의 render 함수로 <div> 태그를 화면이 표시하고 componentDidMount 함수가 실행된다.

5: id가 map인 line 15의 <div> 태그를 mapContainer 변수에 할당한다.

6~9: 표시될 지도의 중심 좌표와 확대 레벨을 할당한다. 중심 좌표는 위도, 경도 순으로 입력한다. 확대 레벨을 숫자가 클수록 넓은 지역을 간단히 보여준다.

10: kakao map api의 Map 함수를 실행하면서 파라미터로 mapContainer와 mapOption를 전달한다.

1: 컴포넌트 안에 kakao 객체가 선언되지 않으므로 line 10에서 kakao가 정의되지 않았다는 에러가 발생한다. 이때 /\*global kakao\*/을 상단에 추가하면 에러를 해결할 수 있다.

cmd 창에서 react 경로(C:\react\_kakaomap)로 이동하고 npm install 명령어를 실행한다.

|  |
| --- |
|  |

**<그림 295>**

npm start 명령어로 react 서버를 실행한다.

|  |
| --- |
|  |

**<그림 296>**

react 서버가 구동되면 다음과 같이 포트번호가 없는 로컬 서버 도메인(localhost)으로 react 페이지를 호출할 수 있다.

|  |
| --- |
|  |

**<그림 297>**

190. 코로나 확진자 정보 excel 파일을 json 파일로 변환하기

**• 학습 내용:** 코로나 확진자 정보가 작성된 excel 파일을 json 형태의 데이터로 변환한다.

**• 힌트 내용:** json 파일을 react 서버 경로에 위치시키고 좌표 정보를 조회에 지도에 표시한다.

웹 사이트에서 수정 빈도가 적은 데이터를 조회하는 용도로만 사용할 것이라면, db 서버를 연결하지 않고 json 파일만 사용할 수 있다. json 파일에 데이터를 작성하고 필요할 때 조회한다. 데이터 수정은 json 파일보다 excel로 하는 것이 편리하기 때문에 excel로 작성한 데이터를 json 형태의 데이터로 변환해 사용한다.

1. 다음과 같이 excel 파일을 열어 코로나 확진자 정보를 확인한다. lat(위도), lng(경도) 좌표는 확진자가 있는 병원의 좌표를 의미한다.

|  |
| --- |
|  |

**<그림 298>**

2. Excel To Json Converter(<http://beautifytools.com/excel-to-json-converter.php>) 웹 사이트를 이용해 간편하게 excel 파일을 json 파일로 변환할 수 있다. 화면 왼쪽 상단의 [Browse] 버튼을 누르면, 다음과 같이 변환할 excel 파일을 선택할 수 있다.

|  |
| --- |
|  |

**<그림 299>**

3. json으로 변환된 텍스트를 그대로 복사해 src 폴더의 map\_info.json 파일에 붙여 넣는다.

<소스> map\_info.json

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20  21  22  23 | {  "positions": [  {  "num": "1",  "sub\_num": "0",  "lat": "37.478682",  "lng": "126.668475",  "date": "1월19일",  "comment": "인천의료원 격리",  "gender": "여",  "birth": "84",  "nation": "중국",  "contact": "45",  "Isolation": "0",  "finish": "완치",  "hospi\_img": "https://img.khan.co.kr/news/.......01000131000005261.jpg",  "finish\_date": "2월 6일",  "via\_path": "(1월 19일) 검역 단계에서 발열 등 증상이 있는 환자를 검역 조사해...  },  {  "num": "2",  "sub\_num": "0",  .. 코드 생략.. |

191. 확진자 위치에 마커 추가하기

**• 학습 내용:** json 파일의 확진자 격리 병원 좌표를 사용해 지도에 마커를 추가한다.

**• 힌트 내용:** react 컴포넌트에서 json 파일을 임포트해 데이터를 조회한다.

react 컴포넌트에서 json 파일 경로를 임포트하면, json 파일에 작성된 데이터를 조회할 수 있다. 확진자 수가 여러 명이기 때문에 반복문을 사용해 확진자 수 만큼 마커를 생성한다.

src 폴더의 map\_info.json 파일을 확인한다.

<소스> map\_info.json

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17 | {  "positions": [  {  "num": "1",  "sub\_num": "0",  "lat": "37.478682",  "lng": "126.668475",  "date": "1월19일",  "comment": "인천의료원 격리",  .. 코드 생략..  },  {  "num": "2",  "sub\_num": "0",  "lat": "37.567496",  "lng": "127.005116",  .. 코드 생략.. |

6, 15: 지도에서 마커가 생성될 확진자 격리 병원의 위도 좌표다.

7, 16: 지도에서 마커가 생성될 확진자 격리 병원의 경도 좌표다.

src 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29 | /\*global kakao\*/  import React, { Component } from 'react';  import jsonData from "./map\_info.json";  class App extends Component {  componentDidMount() {  var mapContainer = document.getElementById('map'),  mapOption = {  center: new kakao.maps.LatLng(37.505496, 127.005116),  level: 9  };  var map = new kakao.maps.Map(mapContainer, mapOption);  for(var i=0 ; i<jsonData.positions.length ; i++){  var position = jsonData.positions[i]  var marker = new kakao.maps.Marker({  map: map,  position : new kakao.maps.LatLng(position.lat, position.lng),  });  }  }  render() {  return (  <div id="map" style={{"width":"100%","height":"700px"}}></div>  )  }  }  export default App; |

3: App.js와 같은 폴더에 위치한 map\_info.json 파일을 임포트해 사용할 수 있도록 한다.

9~10: 지도의 중심 좌표와 확대 레벨을 마커가 표시될 서울 중심으로 변경한다.

14~20: json 파일을 positions 배열에 접근해 배열의 크기(length)만큼 for문을 반복 실행한다.

15: 코드 중복을 최소화하기 위해 반복문이 실행될 때마다 position 변수에 배열 인자를 할당한다.

16: kakao map api의 Marker 함수로 마커를 생성한다. map과 position 변수를 파라미터로 전달한다.

17: 파라미터 map에는 마커가 표시되는 지도인 line 12의 map 변수를 할당한다.

18: 파라미터 position에는 kakao map api의 LatLng 함수를 사용해 마커가 표시될 좌표를 할당한다.

json 파일에 작성된 5명의 확진자가 격리된 병원의 좌표에 마커가 추가된 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 300>**

192. 확진자 위치에 이미지 마커 추가하기

**• 학습 내용:** 필요한 이미지를 서버 경로에 업로드한 후 마커마다 각각 다른 이미지를 표시한다.

**• 힌트 내용:** 확진자 번호와 완치 여부에 따라 다른 이미지를 표시한다.

마커가 모두 동일한 이미지로 표시된다면, 어떤 확진자의 위치인지 지도에서 확인할 수 없다. Kakao map api의 Marker 함수를 호출할 때 이미지 경로를 파라미터로 전달하면 원하는 이미지가 마커로 표시된다.

1. 다음과 같이 [src] – [img] 폴더에 마커로 사용할 이미지를 추가한다. 완치자의 이미지명은 clearp + 확진자 번호.png, 확진자의 이미지명은 hop + 확진자 번호.png 형태로 사용한다.

|  |
| --- |
|  |

**<그림 301>**

src 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34 35  36  37  38  39  40 | /\*global kakao\*/  import React, { Component } from 'react';  import jsonData from "./map\_info.json";  class App extends Component {  componentDidMount() {  var mapContainer = document.getElementById('map'),  mapOption = {  center: new kakao.maps.LatLng(37.505496, 127.005116),  level: 9  }  var map = new kakao.maps.Map(mapContainer, mapOption)  for(var i=0 ; i<jsonData.positions.length ; i++){  var position = jsonData.positions[i]  var imageSrc = require("./img/hop0"+(i+1)+".png")  if(position.finish == '완치'){  imageSrc = require("./img/clearp0"+(i+1)+".png")  }  var imageSize = new kakao.maps.Size(60, 70)  var imageOption = {offset: new kakao.maps.Point(27, 69)}  var markerImage = new kakao.maps.MarkerImage(imageSrc, imageSize, imageOption)  var marker = new kakao.maps.Marker({  map: map,  position : new kakao.maps.LatLng(position.lat, position.lng),  image: markerImage  });  }  }  render() {  return (  <div id="map" style={{"width":"100%","height":"700px"}}></div>  )  }  }  export default App; |

17: map\_info.json의 데이터의 확진자 번호 순서로 작성돼 있다. for문의 i 변수에 1을 더한 값과 확진자 번호가 일치한다. 사용할 확진자 이미지 경로를 require해 imageSrc 변수에 할당한다.

18~20: 배열 인자 중 finish 변수가 완치인 경우, 완치자 이미지 경로를 imageSrc 변수에 할당한다.

22: Size 함수로 마커 이미지의 크기를 지정할 수 있다. 너비(60)와 높이(70) 순서로 파라미터를 전달한다. 함수 실행 결과를 imageSize 변수에 할당한다.

23: Point 함수를 사용하면 마커 좌표를 기준으로, 이미지를 표시할 위치를 설정할 수 있다. 가로(27)와 세로(69) 순서로 파라미터를 전달한다. 가로 값이 커지면 왼쪽으로, 세로 값이 커지면 위쪽으로 이미지가 이동한다. 함수 실행 결과를 imageOption 변수에 할당한다.

25: MarkerImage 함수로 설정한 마커 이미지 정보를 파라미터로 전달한다. 함수 실행 결과를 markerImage 변수에 할당한다.

29: 마커를 생성하는 Marker 함수를 호출할 때 image 변수에 markerImage 변숫값을 할당해 전달한다.

이미지를 추가하고 코드를 실행해보면, 다음과 같이 확진자 번호와 완치 여부가 이미지로 표시된다.

|  |
| --- |
|  |

**<그림 302>**

193. 마커에 클러스터러 사용하기

**• 학습 내용:** 클러스터러를 적용해 지도 축소 시 단위 면적당 확진자 수를 확인한다.

**• 힌트 내용:** 클러스터러를 사용하면, 지역별 확진자 수를 한눈에 확인할 수 있다.

지도를 축소할수록 많은 마커가 화면에 표시된다. 한 화면에 너무 많은 마커가 표시되면, 가독성이 떨어진다. 클러스터러를 사용하면 지역별 마커 수를 표시하기 때문에 넓은 면적이 데이터를 확인할 때 유용하다.

jquery의 map 함수를 사용하려면 jquery 패키지를 설치해야 한다. cmd 창을 열어 react 경로(C:\reactMapKakao)로 이동한다. [npm install --save jquery]를 입력하면, 다음과 같이 npm이 jquery을 설치한다.

|  |
| --- |
|  |

**<그림 303>**

public 폴더의 index.html 파일을 확인한다.

<소스> index.html

|  |  |
| --- | --- |
| 1 2  3  4  5  6  8 9  10  11  12  13 | <!DOCTYPE html>  <html lang="en">  <head>  .. 코드 생략..  <title>React App</title>  <script type="text/javascript" src=".. 코드 생략..?appkey=[키입력]&libraries=clusterer">  </script>  </head>  <body>  .. 코드 생략..  </body>  </html> |

6: 클러스터러를 사용하기 위해 &libraries=clusterer를 url에 추가한다.

src 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 | .. 코드 생략..  import $ from 'jquery';  class App extends Component {  componentDidMount() {  .. 코드 생략..  var clusterer = new kakao.maps.MarkerClusterer({  map: map,  averageCenter: true,  minLevel: 5  });  var markers = $(jsonData.positions).map(function(i, position) {  var imageSrc = require("./img/hop0"+(i+1)+".png")  if(position.finish == '완치'){  imageSrc = require("./img/clearp0"+(i+1)+".png")  }  var imageSize = new kakao.maps.Size(60, 70)  var imageOption = {offset: new kakao.maps.Point(27, 69)}  var markerImage = new kakao.maps.MarkerImage(imageSrc, imageSize, imageOption)  var marker = new kakao.maps.Marker({  map: map,  position : new kakao.maps.LatLng(position.lat, position.lng),  image: markerImage  });  return marker;  })  clusterer.addMarkers(markers);  }  .. 코드 생략.. |

2: line 13의 map 함수를 사용하기 위해 jquery를 임포트한다.

7: 클러스터러를 사용하기 위해 MarkerClusterer 함수에 클러스터링 설정 변수들을 파라미터로 전달한다. 함수 실행 결과를 clusterer 변수에 할당한다. 파라미터 map에는 클러스터러를 적용한 지도 변수를 할당한다. averageCenter가 true면 클러스터링되는 마커들의 중간 좌표에 클러스터링 마커가 표시된다. minLevel은 값이 클수록 지도를 더 많이 축소해야 클러스터링 기능이 적용된다.

13: for문 대신 jquery map 함수를 사용해 반복문을 실행한다. Line 30에서 반환된 marker 변수들을 markers 변수에 할당한다.

30: 마커들의 정보가 할당돼 있는 markers 변수를 clusterer.addMarkers 함수의 파라미터로 전달하면 지도에 클러스터러가 적용된다.

지도를 축소할수록 클러스터링 되는 마커들이 많아지는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 304>**

194. 확진자 정보 커스텀 오버레이로 표시하기

**• 학습 내용:** 지도에서 마커를 클릭하면, 커스텀 오버레이를 표시해 확진자 정보를 표시한다.

**• 힌트 내용:** 커스텀 오버레이를 사용하면, 원하는 디자인의 팝업을 표시할 수 있다.

커스텀 오버레이를 사용하면, 원하는 형태의 팝업을 html과 css로 구현할 수 있다. 클릭한 마커에 관련된 정보를 제공할 때 사용하고 마커 좌표를 기준으로 상대 좌표를 적용해 팝업을 표시할 수 있다.

src 폴더의 App.css 파일에 커스텀 오버레이 관련 코드를 추가한다.

<소스> App.css

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7 | .. 코드 생략..  .overlaybox li .up {background-position:0 -40px;}  .overlaybox li .down {background-position:0 -60px;}  .overlaybox li .count {position:absolute;margin-top:5px;right:15px;font-size:10px;}  .overlaybox li:hover {color:#fff;background:#d24545;}  .overlaybox li:hover .up {background-position:0 0px;}  .overlaybox li:hover .down {background-position:0 -20px;} |

src 폴더의 App.js 파일을 확인한다.

<소스> App.js

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 9  10  11  12  13  14 15  16  17  18  19  20 21  22  23  24  25  26 27  28  29  30  31  32 33  34 35  36  37  38  39  40  41  42  43 44 45  46 47  48  49  50  51  52 53  54 55  56  57  58  59  60  61  62  63 | .. 코드 생략..  import './App.css';  class App extends Component {  componentDidMount() {  .. 코드 생략..  var content = '<div class="overlaybox">' +  '<div class="boxtitle">'+position.num+'번째 확진자</div>' +  '<div class="boxtitle2" style="margin-left: 150px;">'+  '<font color="white"">[닫기] 버튼 [X]</font></div>'+  '<div class="first first\_'+position.num+'">' +  ' <div class="triangle text">'+position.num+'</div>' +  ' <div class="movietitle text">'+position.comment+'</div>' +  '</div>' +  '<ul style = "overflow:scroll">' +  ' <li class="up">' +  ' <span class="number">격리:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.date+'</span>' +  ' <span class="number">완치:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.finish\_date+'</span>' +  ' </li>' +  ' <li class="up">' +  ' <span class="number">성별:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.gender+'</span>' +  ' <span class="number">생년:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.birth+'</span>' +  ' <span class="number">국적:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.nation+'</span>' +  ' </li>' +  ' <li class="up">' +  ' <span class="number">접속자수:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.contact+'</span>' +  ' <span class="number">격리조치중:</span>' +  ' <span class="title" style="font-weight:bold;color:white">'+position.Isolation+'</span>' +  ' </li>' +  '</ul>' +  '</div>';    var lat = Number(position.lat)  var lng = Number(position.lng)  var lat\_string = lat.toString()  var lng\_string = lng.toString()    var customOverlay = new kakao.maps.CustomOverlay({  position: new kakao.maps.LatLng(lat\_string, lng\_string),  content: content,  xAnchor: 0.25,  yAnchor: 0.95  });  var clickHandler1 = function(event) {  customOverlay.setMap(map);  $(".boxtitle2").click(function(){  customOverlay.setMap(null);  });  $(".first\_"+position.num).css({"background":"url("+position.hospi\_img+")",  "background-size" :"247px 247px"});  };  kakao.maps.event.addListener(marker, 'click', clickHandler1)  .. 코드 생략.. |

2: App.css를 임포트해 커스텀 오버레이에 스타일이 적용되도록 한다.

7~37: 커스텀 오버레이 팝업에 표시될 html 코드를 생성한다. json 데이터가 할당된 position 변수를 사용해 표시할 데이터를 <html> 태그 사이에 삽입한다. 완성된 html 코드를 content 변수에 할당한다.

39~42: 마커의 좌표를 기준으로 커스텀 오버레이 팝업을 표시해야 하기 때문에 마커 좌표를 문자열로 변환한다.

44~49: CustomOverlay 함수로 커스텀 오버레이를 생성하고 관련 옵션을 파라미터로 전달한다. 파라미터 position에는 커스텀 오버레가 표시될 좌표가 할당된다. content는 line 7에서 생성한 html 코드이다. xAnchor와 yAnchor는 각각 팝업의, X, Y축의 위치를 의미한다. 0에서 1사이의 값을 갖고 기본값은 0.5이다.

62: 마커에 클릭 이벤트가 발생하면, clickHandler1 함수를 실행한다.

52: 클릭한 마커의 커스텀 오버레이 팝업 창을 지도에 표시한다.

54~56: 팝업 창에서 [닫기] 버튼을 누르면, 커스텀 오버레이를 지도에서 표시하지 않는다.

58~59: json 데이터에 있는 확진자별 격리 병원 이미지 경로를 background 속성으로 적용한다. 이미지를 클릭한 마커의, 확진자 번호에 맞게 팝업 창에 표시한다.

마커를 클릭하면 커스텀 오버레이 팝업 창을 띄우고 확진자 번호에 맞는 데이터가 표시된다.

|  |
| --- |
|  |

**<그림 305>**

195. EC2 서버에 로컬 react 소스 옮기기

**• 학습 내용:** 로컬에서 개발한 kakao map 소스를 EC2 서버로 옮긴다.

**• 힌트 내용:** 소스를 tar 파일로 압축하고 SFTP로 EC2 서버에 전송한다.

백엔드 서버를 사용하지 않고 react 서버만 구동하기 때문에 react 경로의 소스만 업로드한다. 패키지 파일들이 설치된 node\_modules 폴더를 제외하고 tar 파일로 압축해 SFTP로 전송한다. 압축된 파일은 SSH에서 명령어를 실행해 압축을 푼다.

|  |
| --- |
| NOTE |
| EC2 서버는 기존에 react, node 서버를 구동했던 인스턴스를 사용한다. 기존 서버는 3000번(react), 5000번(node)포트를 사용했다. kakao map을 구현한 react 서버는 80포트를 사용할 것이기 때문에 포트가 겹치지 않아 기존 인스턴스에서 서버 구동이 가능하다. 물론 node와 npm도 기존에 설치했기 때문에 추가로 설치하지 않는다. |

1. 로컬 react 경로(C:\reactMapKakao)에서 node\_modules 폴더를 제외하고 tar 파일로 압축한다.

|  |
| --- |
|  |

**<그림 306>**

2. SFTP로 EC2 서버에 접속한 후 압축을 풀 폴더(/home/ubuntu/reactMapKakao/)를 생성한다. 압축한 tar 파일을 생성한 폴더에 업로드한다.

|  |
| --- |
|  |

**<그림 307>**

3. SSH로 EC2 서버에 접속한다. root 계정으로 전환[sudo su –]한 후 압축을 푼 경로로 이동한다[(/home/ubuntu/reactMapKakao/]. 그리고 업로드한 tar 파일의 압축을 해제한다[reactMapKakao.tar].

|  |
| --- |
|  |

**<그림 308>**

4. EC2 서버에 연결된 SFTP 창에서 폴더를 새로 고침하면, 다음과 같이 tar 파일의 압축이 풀려 있는 것을 확인할 수 있다.

|  |
| --- |
|  |

**<그림 309>**

196. EC2 서버에 react 패키지 설치하기

**• 학습 내용:** react 서버 구동에 필요한 패키지들을 압축을 푼 EC2 서버 경로에 설치한다.

**• 힌트 내용:** npm 명령어로 package.json 파일에 작성돼 있는 react 패키지들을 설치한다.

이전 예제에서 react, node 서버를 EC2 서버에 배포했다. 이때는 react 경로와 node 경로에서 각각 패키지를 설치했다. kakao map 프로젝트는 react 서버만 존재하기 때문에 react 경로에서만 패키지를 설치하면 된다.

1. SSH로 EC2 서버 터미널에 접속한다. root 계정으로 전환[sudo su –]한 후 react 경로로 이동한다[cd /home/ubuntu/reactMapKakao/]. 그리고 package.json에 기록돼 있는 react 패키지들을 [npm install] 명령어로 설치한다.

|  |
| --- |
|  |

**<그림 310>**

|  |
| --- |
| NOTE |
| [npm install] 명령어를 실행하는데 특정 패키지에서 node 버전이 너무 낮아 설치할 수 없다는 에러가 발생할 수 있다. 이런 경우 다음과 같은 명령어로 node를 재설치한다.  1. [apt-get --purge remove nodejs] 명령어로 nodejs를 삭제한다.  2. [curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -] 명령어로 설치를 원하는 버전의 node 버전(setup\_14.x)을 입력한다. setup\_14.x는 앞자리가 14인 nodejs 버전을 의미한다.  3. [apt-get install -y nodejs] 명령어로 nodejs를 다시 설치한다. |

2. SFTP로 EC2 서버에 접속한 후 react 경로로 이동한다[cd /home/ubuntu/reactMapKakao/]. 다음과 같이 node\_modules 폴더가 생성된 것을 확인할 수 있다.

|  |
| --- |
| **<그림 311>** |

197. EC2 서버에 80포트 설정하기

**• 학습 내용:** EC2 react 서버를 포트 없이 사용하기 위해 80포트 설정을 한다.

**• 힌트 내용:** 로컬에서 이미 80포트 설정을 해줬지만, ubuntu에서는 다른 명령어가 사용된다.

로컬(윈도우 운영체제)에서 react 서버를 포트 없이 구동하기 위해 package.json 파일에 80포트 설정 코드를 추가했다. ubuntu에서는 포트 설정 명령어가 다르기 때문에 코드를 수정해 EC2 서버에 업로드해야 한다. 또 EC2 서버의 인바운드 포트 80번을 추가해 외부에서 접근할 수 있도록 한다.

1. 로컬 react 경로(C:\reactMapKakao)의 package.json 파일을 다음과 같이 수정한다.

<소스> package.json

|  |  |
| --- | --- |
| 1 2  3  4  5  6  7  8 | .. 코드 생략..  "scripts": {  "start": "export PORT=80 && react-scripts start",  "build": "react-scripts build",  "test": "react-scripts test",  "eject": "react-scripts eject"  },  .. 코드 생략.. |

3: 윈도우에서 사용하던 [set PORT=80] 명령어를 ubuntu에서 사용하는 [export PORT=80]로 수정한다.

2. SSH로 EC2 서버 터미널에 접속한다. root 계정으로 전환[sudo su –]한 후 react 경로의 상위 폴더로 이동한다[cd /home/ubuntu/]. 그리고 SFTP에서 ubuntu 계정으로 파일을 업로드할 수 있도록 group 권한을 ubuntu 계정에 부여한다. [chown -R root:ubuntu reactMapKakao] 명령어로 group 권한 소유자를 ubuntu로 지정한다. [chmod -R 770 reactMapKakao] 명령어로 group권한 소유자에 읽기, 쓰기, 실행 권한을 부여한다.

|  |
| --- |
|  |

**<그림 312>**

3. SFTP로 EC2 서버에 접속한 후 react 경로로 이동한다[cd /home/ubuntu/reactMapKakao/]. 다음과 같이 group 권한이 ubuntu 계정에 부여된 것을 확인할 수 있다. 로컬에서 수정한 package.json 파일을 SFTP 경로로 드래그해 업로드한다.

|  |
| --- |
|  |

**<그림 313>**

4. EC2 서버로 들어오는 포트 번호를 허용하기 위해 AWS 웹 사이트(<https://aws.amazon.com/ko/>)에 접속한다. 가입한 계정으로 로그인을 하고 [내 계정] - [AWS Management Console]을 클릭한다. 왼쪽 상단의 [서비스] - [컴퓨팅] - [EC2]를 클릭한다. 다음과 같이 왼쪽 메뉴 중 [보안 그룹] 버튼을 누른다. 보안 그룹 이름이 [launch-wizard]로 시작하는 보안 그룹 ID를 클릭한다. 하단에 인바운드 규칙 영역이 표시되면 [인바운드 규칙 편집] 버튼을 누른다.

|  |
| --- |
| NOTE |
| 인바운드 규칙 중 3000번 포트는 지난 EC2 인스턴스 생성 예제에서 추가됐다. |

|  |
| --- |
|  |

**<그림 314>**

4. 다음과 같이 80번 포트를 등록하고 [규칙 저장] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 315>**

198. 카카오 웹 사이트 도메인에 EC2 IP 추가하기

**• 학습 내용:** EC2 react 서버에서 kakao map api를 사용하기 위해 웹 사이트 도메인을 추가한다.

**• 힌트 내용:** 카카오 개발자 웹 사이트에 접속해 웹 사이트 도메인에 EC2 ip를 추가한다.

지난 예제에서 로컬 서버의 접속을 kakao map api가 허용하게 하는 작업으로, 웹 사이트 도메인에 localhost를 추가했다. 마찬가지로 EC2 서버에서 kakao map api를 사용하려면 EC2 서버 ip를 웹 사이트 도메인에 등록해야 한다.

1. 카카오 개발자 웹 사이트 (<https://developers.kakao.com>)에 접속해 카카오 계정으로 로그인한다. 상단의 [내 애플리케이션]을 클릭하고 기존에 추가한 애플리케이션을 선택한다. 그리고 왼쪽 메뉴 중 [플랫폼] 버튼을 누른다. 플랫폼 상세 페이지를 보면, 이전 예제에서 사용한 [localhost]가 [Web] 영역에 등록돼 있다는 것을 알 수 있다. [수정] 버튼을 눌러 다음과 같이 수정 팝업 창이 표시되면, EC2 서버 ip를 입력하고 [저장] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 316>**

199. screen으로 서버 상시 구동하기, 포트로 서버 종료하기

**• 학습 내용:** screen 명령어로 서버를 구동하면, SSH 창을 닫아도 서버가 꺼지지 않는다.

**• 힌트 내용:** screen으로 실행된 서버는 실행 포트를 사용해 종료한다.

지난 react, node 서버를 상시 구동하는 예제에서 nohup이라는 명령어를 사용했다. 간혹 nohup이 제대로 동작하지 않는 경우, 동일한 기능을 하는 screen이라는 명령어를 사용할 수 있다.

1. SSH로 EC2 서버 터미널에 접속한다. root 계정으로 전환[sudo su –]한 후 react 경로로 이동한다[cd /home/ubuntu/reactMapKakao/]. [screen -L npm start] 명령어로 서버를 실행한다.

|  |
| --- |
| NOTE |
| screen은 nohup과 달리, 서버 로그가 SSH 터미널에 출력된다. screen으로 서버를 구동하고 SSH터미널을 닫지 않은 상태에서는 [Ctrl] + [C]를 눌러 서버를 종료시킬 수 있다. nohup의 경우 터미널을 끄지 않은 상태라도 포트 번호를 사용하는 등의 방법으로 서버를 종료해야 한다. |

|  |
| --- |
|  |

**<그림 317>**

2. screen으로 서버를 상시 구동 후 SSH 터미널을 닫고 다시 접속했을 경우, [fuser -k 포트 번호/tcp] 명령어로 서버를 종료할 수 있다.

|  |
| --- |
|  |

**<그림 318>**

200. 도메인 구매 후 ip 연결하기

**• 학습 내용:** 가비아 웹 사이트에서 도메인을 구매한 후 EC2 서버 ip를 연결한다.

**• 힌트 내용:** ip에 비해 도메인을 사용하는 것이 가독성이 좋고 외우기도 쉽다.

EC2 서버는 AWS의 서비스이기 때문에 Amazon Route 53이라는 DNS(도메인 네임 시스템)을 많이 사용한다. Route 53을 사용하면 모니터링, 로드밸런싱 등의 기능을 제공해준다. 그럼에도 불구하고 가비아 웹 사이트를 이용하는 이유는 일단 한국 웹 사이트이기 때문에 결제가 쉽고 UI가 친숙해 눈에 잘 들어오기 때문이다. 특히 문제가 발생했을 때 고객센터에 전화하면 한국인 상담사가 친절하고 빠르게 문제를 해결해준다. AWS의 경우 문제가 생기면 영문 이메일을 여러 번 주고받아야 한다.

|  |
| --- |
| NOTE |
| 도메인이란 웹 브라우저 주소 창에 ip 주소(숫자)대신, 읽고 외우기 쉬운 도메인 주소( 문자)로 웹 사이트에 접속할 수 있도록 하는 서비스다. |

1. 가비아 도메인 웹 사이트(https://domain.gabia.com)에 접속한다. 생성을 원하는 도메인을 검색 창에 입력한 후 [검색] 버튼을 누른다. 검색 결과 중 원하는 도메인을 선택한 후 [신청하기] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 319>**

2. 서비스 신청 페이지에서 신청 정보와 소유자 정보 등 필요한 정보를 입력한 후 [다음 단계] 버튼을 누른다. 이후 단계를 진행해 결제를 완료한다.

|  |
| --- |
|  |

**<그림 320>**

3. 결제 완료 후 가비아 도메인 페이지로 돌아와 [도메인 정보 변경] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 321>**

4. 도메인 관리 페이지에서 왼쪽 메뉴의 [DNS 정보]를 클릭한다. 추가된 도메인명을 확인한 후 [DNS 관리] 버튼을 눌러 DNS 관리 툴로 이동한다.

|  |
| --- |
|  |

**<그림 322>**

5. DNS관리 페이지(dns.gabia.com)에서 수정할 도메인을 확인한 후 [설정] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 323>**

6. 도메인 상세 페이지에서 [레코드 수정] 버튼을 누르면, 다음과 같이 레코드를 추가할 수 있는 팝업이 표시된다. 타입을 A로 선택한 후 생성한 도메인에 ip를 연결해 사용한다. 호스트는 도메인 앞에 붙는 문자열인데, 사용하지 않을 경우 @를 입력한다. 값/위치에는 도메인에 연결할 ip(ec2 서버의 탄력적 ip)를 입력한다. TTL값은 ip에 연결한 도메인이 적용될 때까지 걸리는 시간(초)이다. A 레코드의 경우 기본으로 600(10분)을 설정한다. 입력이 완료되면 [확인], [저장] 버튼을 순서대로 누른다.

|  |
| --- |
| NOTE |
| 레코드: A 레코드란, 도메인이 ip 주소를 참고해 웹 서버에 연결되도록 하는 것이다. ip가 아닌 도메인을 연결하는 경우 CNAME 레코드를 설정한다. CNAME이란, 실제 도메인 이름과 연결되는 가상의 도메인 이름을 의미한다. 호스트: 도메인이 react200.shop이고 호스트가 www인 경우, www.react200.shop으로 호출해야 웹 사이트에 접속할 수 있다.  TTL: TTL 값은 작게 할수록 도메인 전파 속도를 높일 수 있지만, 도메인 서버의 처리 시간을 느리게 할 수 있으므로 권장하는 값으로 설정한다. |

|  |
| --- |
|  |

**<그림 324>**

7. 카카오 개발자 웹 사이트 (<https://developers.kakao.com>)에 접속해 카카오 계정으로 로그인한다. 상단의 [내 애플리케이션]을 클릭한 후 기존에 추가한 애플리케이션을 선택한다. 그리고 왼쪽 메뉴 중 [플랫폼] 버튼을 누른다. 플랫폼 상세 페이지를 보면, 이전 예제에서 사용한 [localhost]와 [ec2 서버 ip]가 [Web] 영역에 등록돼 있다. [수정] 버튼을 눌러 다음과 같이 수정 팝업 창이 표시되면, 생성한 도메인을 입력하고 [저장] 버튼을 누른다.

|  |
| --- |
|  |

**<그림 325>**

8. 도메인 전파 시간(TTL)으로 설정한 10분이 지나고 생성한 도메인을 웹 브라우저에서 호출한다. 다음과 같이 도메인으로 ec2 서버 ip가 연결된 웹 서버에 접속할 수 있다.

|  |
| --- |
|  |

**<그림 326>**