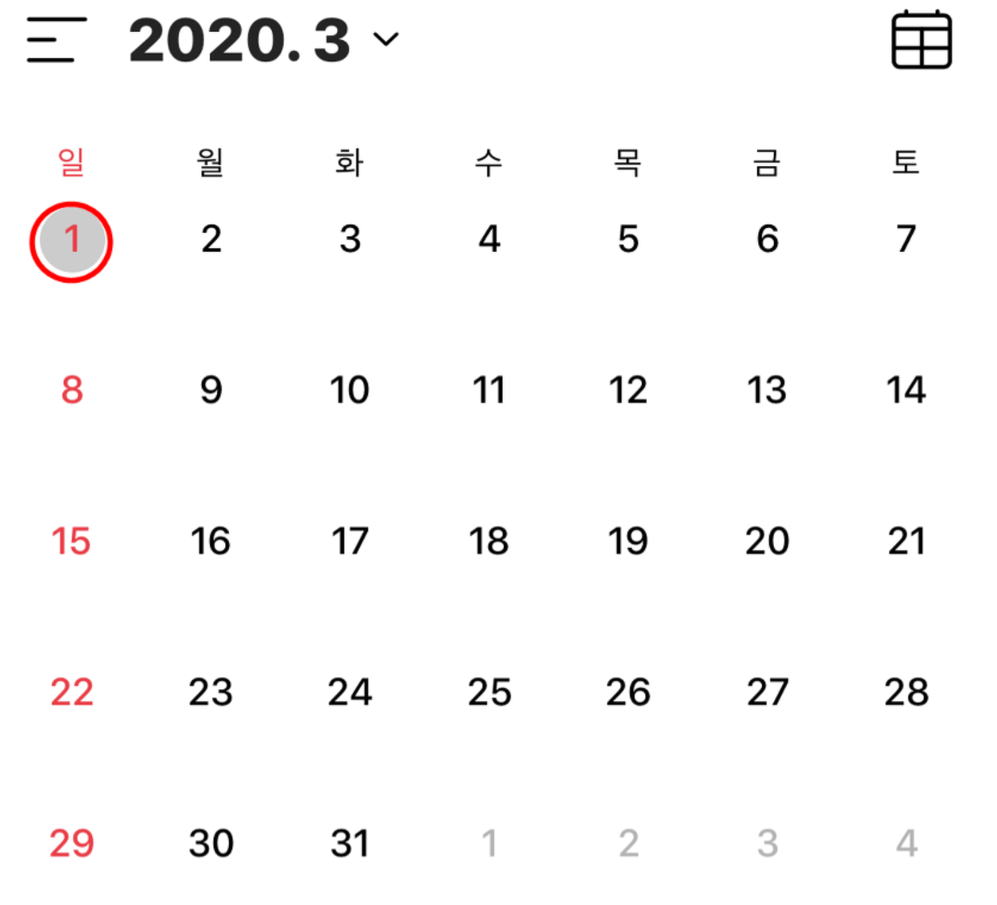
React로 만드는 calendar app만들기

# Calendar 날짜 표현하기

앉아있는, 남자, 옅은, 서있는이(가) 표시된 사진

자동 생성된 설명

금일 만들어 볼 것은 calendar에 날짜를 표시하는 부분입니다. 날짜를 표시하고 최초 마운트 시 오늘에 해당하는 날짜가 검게 표현되어야 합니다. 또한 토요일과 일요일은 평일과 다르게 색상을 표현해야 합니다. 마지막으로 이번 달이 아닌 날이 달력에 표시되게 되면 흰색으로 표현되게 끔 해야합니다.



**Issue 발생 !**

평소 달력은 이렇게 5주가 보통 정상적이지만 달에 따라서는 6주인 경우가 존재하고 2월의 경우 4주가 존재하는 경우가 있습니다. 달력이 만약 5주에서 6주 그리고 4주로 계속해서 레이아웃이 변경 되는 경우 사용자가 불편함을 느낄 가능성이 매우 높습니다. 그렇기에 우리는 5주를 기본 주로 두고, 6주인 경우 23일과 24일에 30과 31일을 같이 표기하겠습니다.

**Issue 발생 !**

좋은 시도라고 생각하였으나, mobile 환경을 고려하였을 때 30일과 31일 컴포넌트가 맨 아래에 존재해야 하는데, 이상한 코드로 강제로 분리하는 방법은 좋지 못하다고 판단되었습니다. 그렇기에 6주를 기본으로 두고 달력을 표시하도록 하겠습니다.

**/src/calendar.js**

…

class Week extends Component {

state = {}

Days = (firstDayFormat) => {

const \_days = [];

for (let i = 0; i < 7; i++) {

const Day = moment(firstDayFormat).add('d', i);

\_days.push({

yearMonthDayFormat: Day.format("YYYY-MM-DD"),

getDay: Day.format('D'),

isHolyDay: false

});

}

return \_days;

}

mapDaysToComponents = (Days, fn = () => { }) => {

return Days.map((dayInfo, i) => {

let className = "date-weekday-label";

if (i === 0) {

className = "date-sun";

} else if (i === 6) {

className = "date-sat"

}

return (

<div className={"RCA-calendar-day " + className} onClick={() => fn(dayInfo.yearMonthDayFormat)}>

<label className="RCA-calendar-day-label">

{dayInfo.getDay}

</label>

{*/\** <label className="RCA-calendar-day">{dayInfo.getDay}</label> *\*/*}

</div>

)

})

}

render() {

return (

<div className="RCA-calendar-week">

{this.mapDaysToComponents(this.Days(this.props.firstDayOfThisWeekformat))}

</div>

)

}

}

…

기본 구상은 6개의 Week을 만들고 Week component에서 다시 7개의 Day Component를 만드는 것입니다. 최대한 하위 컴포넌트에서 함부로 상위 컴포넌트의 객체를 건들 수 없도록 format으로 전달하였기에 코드가 조금 깁니다.

Days = (firstDayFormat) => {

const \_days = [];

for (let i = 0; i < 7; i++) {

const Day = moment(firstDayFormat).add('d', i);

\_days.push({

yearMonthDayFormat: Day.format("YYYY-MM-DD"),

getDay: Day.format('D'),

isHolyDay: false

});

}

return \_days;

}

Week.Days 는 파라메터로 이번 주 첫날에 대한 데이터를 받고, 각 Day컴포넌트에 해당하는 날짜 정보를 가진 moment객체를 만들어 \_days 배열에 넣어 반환하게 됩니다. 일주일의 7일은 Static한 부분 임으로 7번의 반복을 실행하여도 무관 합니다.

mapDaysToComponents = (Days, fn = () => { }) => {

return Days.map((dayInfo, i) => {

let className = "date-weekday-label";

if (i === 0) {

className = "date-sun";

} else if (i === 6) {

className = "date-sat"

}

return (

<div className={"RCA-calendar-day " + className} onClick={() => fn(dayInfo.yearMonthDayFormat)}>

<label className="RCA-calendar-day-label">

{dayInfo.getDay}

</label>

{*/\** <label className="RCA-calendar-day">{dayInfo.getDay}</label> *\*/*}

</div>

)

})

}

이후 등장하게 되는 Week.mapDaysToComponents은 첫번째 파라메터로 모멘트 객체로 이루어진 배열을 받게 되며, 해당 배열을 map 메서드를 이용하여 Day component를 7개 만들게 됩니다. 일요일과 토요일은 평일과 다르게 다른 컬러로 색상을 표시해야 하기 때문에, day component의 index 순서에 따라 표시에 따른 구분을 해줍니다.

2번째 parametar로 콜백을 받는데 이는 컴포넌트를 클릭하게 되면 컴포넌트의 색상을 변경하고 다른 추가 목적을 위해서도 사용할 수 있게끔 의도한 것입니다.

**/src/calendar.js**

export default class Calendar extends Component {

Weeks = (monthYear) => {

const firstDayOfMonth = moment(monthYear).startOf('month');

const firstDateOfMonth = firstDayOfMonth.get('d');

const firstDayOfWeek = firstDayOfMonth.clone().add('d', -firstDateOfMonth);

*// const lastDayOfThisCalendar = dayOfThisCalendar.clone().add('d', 6 \* 7);*

const \_Weeks = [];

for (let i = 0; i < 6; i++) {

\_Weeks.push((

<Week key={`RCA-calendar-week-${i}`} firstDayOfThisWeekformat={firstDayOfWeek.clone().add('d', i \* 7).format("YYYY-MM-DD")} />

))

}

return \_Weeks

}

render() {

return (

<div className="RCA-calendar-container">

<DateHeader dates={"Sun, Mon, Tue, Wed, Thu, Fri, Sat"} />

{this.Weeks(this.props.YM)}

</div>

)

}

}

Calendar container는 6개의 Week component을 만들어 각 주의 일요일을 props로 전달해야만 합니다. 그러기 위해서는 이번 달에 대한 정보를 파라메터로 제공받고 이를 이용하여, 1일의 시작과 요일을 알아야만 합니다.

const firstDayOfMonth = moment(monthYear).startOf('month');

const firstDateOfMonth = firstDayOfMonth.get('d');

const firstDayOfWeek = firstDayOfMonth.clone().add('d', -firstDateOfMonth);

moment객체가 제공하는 startOf 메서드는 모멘트 객체의 연도, 월의 처음 날로 moment객체의 정보를 변환 시켜 줍니다. 그리고 필요한 것은 해당 월의 1일이 어떤 요일에 속해있는지 입니다. 이는 get(‘d’) 메서드를 이용해 쉽게 알 수 있습니다. 일요일부터 토요일까지 각 요일은 0 ~ 6까지 대응 되게 됩니다. 그리고 월의 첫날에서 요일만큼 뒤로 가면 숨겨진 전(prev)달의 마지막 일요일의 날을 구할 수가 있게 됩니다.

예를 들어 9월 1일은 화요일이므로 firstDateOfMonth는 2가 됩니다. 그러므로 첫날에서 2일만큼 뒤로 가게 되면 전달의 마지막 일요일이 8월 30일에 대한 정보가 firstDayOfWeek에 저장되게 됩니다.

const \_Weeks = [];

for (let i = 0; i < 6; i++) {

\_Weeks.push((

<Week key={`RCA-calendar-week-${i}`} firstDayOfThisWeekformat={firstDayOfWeek.clone().add('d', i \* 7).format("YYYY-MM-DD")} />

))

}

return \_Weeks

}

우리는 모든 달력이 6주로 표시되길 바라니 6번의 반복문을 사용합니다. 그리고 이번 달력의 최초 일요일부터 일주일 즉 7씩 증가하게 되면 첫 주에서 7일 뒤 일요일, 2주 뒤 일요일, 3주 뒤 일요일 … 6주 째 일요일의 일을 알 수 있습니다 .이를 이용하여 각 주의 첫번째 주를 계산하고 \_Weeks 배열에 추가 해줍니다.

**/src/app.js**

render() {

return (

<div className="test-layout">

<div className="RCA-app-container">

<Header calendarYM={this.state.calendarYM.format("YYYY년 MM월")}

today={this.state.today.format("현재 YYYY - MM - DD")}

moveMonth={this.moveMonth}

/>

<Calendar YM={this.state.calendarYM.format("YYYY-MM-DD")}/>

</div>

</div>

)

}

우리가 이 달력의 달은 App 컴포넌트의 state.calendarYM에서 관리합니다. 그러므로 Calendar에 이 월에 대한 정보를 전달하도록 합니다.

**/src/Calendar.js Calendar.js render내부**

render() {

return (

<div className="RCA-calendar-container">

<DateHeader dates={"Sun, Mon, Tue, Wed, Thu, Fri, Sat"} />

{this.Weeks(this.props.YM)}

</div>

)

}

전달받은 YM 내용은 string이니 Weeks메서드에 전달인자로 제공하고 호출 하도록 합니다 .

**/src/RCA.css**

.RCA-calendar-day-container{

flex-grow: 1;

display : flex;

flex-wrap: wrap;

}

.RCA-calendar-day{

box-sizing: border-box;

position: relative;

flex-basis: 14.2857143%;

border: 0.1px solid white;

border-top: none;

background-color: rgb(225,224,224);

}

.RCA-calendar-week{

display: flex;

flex-grow: 1;

}

.RCA-calendar-day:nth-child(odd){

border-left : none;

border-right: none;

}

.RCA-calendar-day-label{

display: inline-block;

font-size: 1.2rem;

position: absolute;

text-align: center;

line-height: 25px;

top : 5px;

right: 5px;

}

.date-weekday-label{

color : rgb(71, 71, 71)

}

스타일 까지 적용하고 나면 다음과 같은 결과를 확인 할 수 있습니다.

덮여있는, 앉아있는, 오렌지, 긴이(가) 표시된 사진

자동 생성된 설명

여기 까지 되었으면 이제 이번 달과 이번 달이 아닌 달의 label의 색상을 구별 지어야 합니다.

**/src/Calendar.js Calendar.Weeks 메서드 내부**

for (let i = 0; i < 6; i++) {

\_Weeks.push((

<Week key={`RCA-calendar-week-${i}`}

ymOfThisCalendar={firstDayOfMonth.format("YYYY-MM")}

firstDayOfThisWeekformat={firstDayOfWeek.clone().add('d', i \* 7).format("YYYY-MM-DD")} />

))

}

이번 달의 정보를 Week 컴포넌트에 props로 전달하여 줍니다.

**/src/Calendar.js Week.mapDayToComponents 메서드 내부**

mapDaysToComponents = (Days,calendarMonthYear ,fn = () => { }) => {

const thisMonth = moment(calendarMonthYear);

return Days.map((dayInfo, i) => {

let className = "date-weekday-label";

if (!thisMonth.isSame(dayInfo.yearMonthDayFormat,'month')) {

className = "date-notThisMonth";

} else if (i === 0) {

className = "date-sun"

}else if(i===6){

className ="date-sat"

}

return (…)

}}

메서드 파라메터로 이번 달에 대한 정보를 추가로 받도록 합니다. 그리고 전달 받은 이번 달 정보로 moment객체를 만들어줍니다. If문의 조건을 변경하여 thisMonth객체가 만드는 day component의 연도와 월이 같은지를 확인합니다. (isSame 메서드의 month체크 시 연도 월이 같아야 true를 반환합니다.)

만약 다르다면 이 day component가 이번 달이 아니라고 className으로 지정합니다.

**/src/calendar.js Week render메서드 내부**

render() {

return (

<div className="RCA-calendar-week">

{this.mapDaysToComponents(this.Days(this.props.firstDayOfThisWeekformat),this.props.ymOfThisCalendar)}

</div>

)

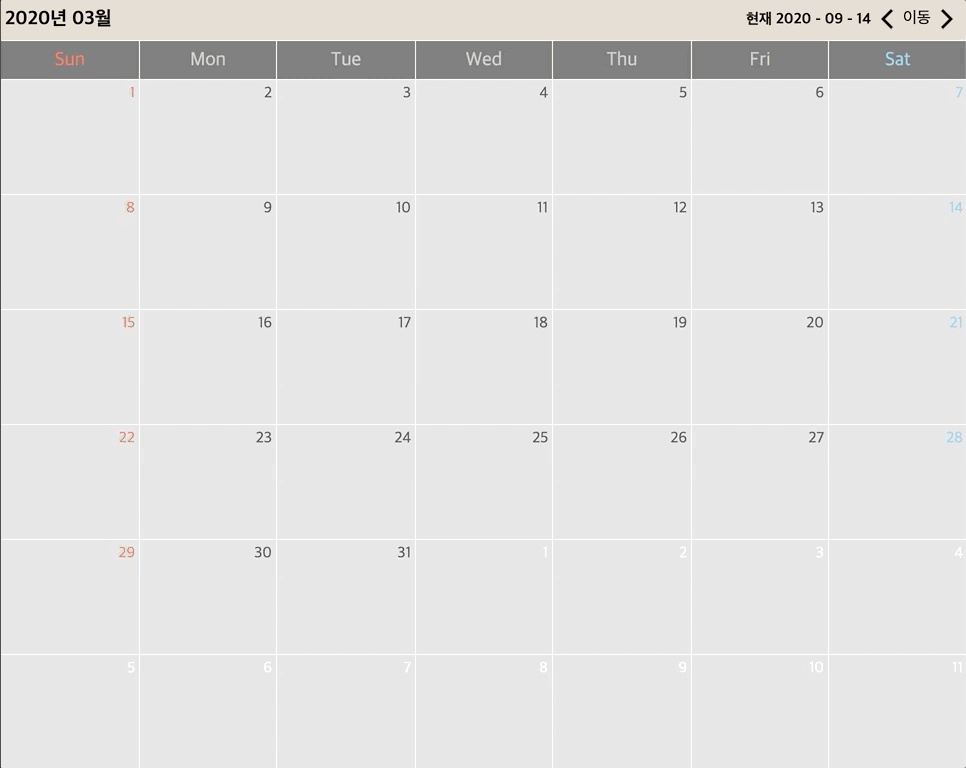
}

mapDaysToComponent의 두번째 인자로 프롭스로 전달받은 연도, 월 정보를 전달합니다.

**결과**

**사진, 앉아있는, 덮여있는, 테이블이(가) 표시된 사진

자동 생성된 설명**

****