The Road To Learn React

مقدمات یادگیری ری‌اکت

نویسنده: روبین ویروش

مترجم: زهره زارعی‌نژاد

ویراستار: سید علی‌اصغر رئیس‌زاده

فهرست مطالب

مقدمه

کتاب «یادگیری مقدمات ری‌اکت» به شما مبانی ری‌اکت را می آموزد. با یادگیری مقدمات ری‌اکت بدون نیاز به ابزارهای پیچیده، یک اپلیکیشن واقعی را ایجاد خواهید کرد. در این کتاب همه چیز از راه‌اندازی تا استقرار بر روی سرور برای شما توضیح داده خواهد شد. این کتاب با منابع و تمرینات اضافی در هر فصل همراه است. پس از خواندن این کتاب می‌توانید اپلیکیشن‌های خود را در ری‌اکت ایجاد کنید. مطالب توسط نویسنده و انجمن به‌روز نگه داشته می‌شود.

با یادگیری مقدمات ری‌اکت می‌خواهم پیش از ورود به اکوسیستم گسترده‌تر ری‌اکت، مبانی کار را ارائه دهم. این مبانی ابزار و مدیریت خارجی state کمتری دارد اما اطلاعات زیادی درمورد ری‌اکت ارائه می‌دهد. این کتاب مفاهیم کلی، الگوها و بهترین شیوه‌ها را در دنیای واقعی ری‌اکت توضیح می‌دهد.

شما یاد خواهید گرفت که اپلیکیشن ری‌اکت خودتان را ایجاد کنید. این یادگیری، ویژگی‌های دنیای واقعی مانند صفحه‌بندی، ذخیره‌سازی سمت سرور و تعاملات دیگر مثل جست‌وجو و مرتب‌سازی را دربرمی‌گیرد. علاوه براین، از جاوااسکریپت ES5 به جاوااسکریپت ES6 منتقل می‌شوید. امیدوارم کتاب جدید، اشتیاق من در مورد ری‌اکت و جاوااسکریپت را نشان دهد و به شما کمک کند تا شروع به کار کنید.

درباره‌ی نویسنده

«رابین ویروش» (Robin wieruch) یک مهندس نرم‌افزار و وب اهل کشور آلمان است که به‌صورت اختصاصی به آموزش برنامه‌نویسی جاوااسکریپت می‌پردازد. او پس از فارغ‌التحصیلی از دانشگاه با مدرک کارشناسی‌ارشد در علوم رایانه هرگز یادگیری خود را متوقف نکرده است. تجربیات ویروش در دنیای استارتاپ، جایی که اوقات فراغت خود را با جاوااسکریپت سپری می‌کرد، به او فرصتی داد تا این موضوعات را به دیگران آموزش دهد.

امروزه ویروش به صورت آزاد مشغول آموزش به دیگران است. برای او تلاش دانش‌آموزان برای پیشرفت از طریق اهداف روشن و یک حلقه‌ی کوتاه بازخورد، یک فعالیت رضایت‌بخش است. این چیزی است که شما در بازخوردهای محل کار خود یاد می‌گیرید. درست است؟ اما خود او هم بدون کدنویسی قادر به آموزش صحیح نخواهد بود. به همین دلیل ویروش زمان باقی‌مانده‌ی خود را به برنامه‌نویسی اختصاص داده است.

چگونه این کتاب را بخوانیم

این کتاب تلاش من برای آموزش نحوه‌ی نوشتن یک اپلیکیشن به شماست. این یک راهنمای عملی برای یادگیری «ری‌اکت» است، و نه یک کتاب مرجع در مورد ری‌اکت. با استفاده از این کتاب می‌توانید یک اپلیکیشن Hacker News بنویسید که با APIهای جهان واقعی در تعامل است. در میان تمام موضوعات جالبی که وجود دارند، در این کتاب مدیریت «استیت» در ری‌اکت، ذخیره و تعامل (مرتب‌سازی و جست‌وجو) پوشش داده می‌شوند. در این مسیر بهترین تمرین‌ها و الگوهای ری‌اکت را آموزش می‌بینید.

علاوه بر این، این کتاب به شما امکان گذر از JavaScript ES5 به JavaScript ES6 را می‌دهد. ری‌اکت بسیاری از ویژگی‌های JavaScript ES6 را شامل می‌شود، و می‌خواهم به شما نشان دهم چگونه می‌توانید از آن‌ها استفاده کنید.

به‌طورکلی، هر گفتار از کتاب بر پایه‌ی گفتار قبل از آن نوشته شده است. هر گفتار به شما چیز جدیدی یاد می‌دهد. برای خواندن این کتاب عجله نکنید. باید هر مرحله را به‌طور کامل یاد بگیرید. می‌توانید برداشت‌های خود را داشته باشید و در مورد هر موضوع بیش‌تر مطالعه کنید. در پایان هر گفتار به شما چند تمرین و مطالب بیش‌تر برای مطالعه داده خواهد شد. اگر واقعاً می‌خواهید ری‌اکت را یاد بگیرید به‌شدت توصیه می‌کنم مطالب اضافی را بخوانید و تمرین‌ها را انجام دهید. پس از خواندن هر گفتار و پیش از ورود به گفتار بعد، سعی کنید تمام مطلب را یاد گرفته باشید.

در آخر کار، شما تولید یک اپلیکیشن در ری‌اکت را یاد خواهید گرفت. بسیار علاقه‌مندم نتایج کار شما را مشاهده کنم، پس لطفاً زمانی که کتاب را به پایان رساندید، نظر خود را برای من بنویسید. گفتار آخر کتاب گزینه‌های متنوعی را به شما می‌دهد تا به سفر خود در یادگیری ری‌اکت ادامه دهید. در مجموع در وب‌سایت شخصی من[1] می‌توانید مطالب زیادی را در مورد ری‌اکت پیدا کنید.

از آن‌جا که در حال خواندن این کتاب هستید، حدس می‌زنم تازه با ری‌اکت آشنا شده‌اید. این عالی‌ست. در پایان امیدوارم بازخورد شما برای بهبود این مطالب را دریافت کنم تا همه بتوانند ری‌اکت را یاد بگیرند. شما می‌توانید مستقیماً در GitHub[2] یا Twitter[3] با من در تماس باشید.

[1] https://www.robinwieruch.de
[2] https://github.com/the-road-to-learn-react/the-road-to-learn-react
[3] https://twitter.com/rwieruch

مقدمه‌ای بر ری‌اکت

این گفتار مقدمه‌ای در مورد ری‌اکت به شما می‌دهد. ممکن است از خودتان بپرسید: چرا اصلاً باید ری‌اکت را یاد بگیرم؟ این گفتار ممکن است انتقالخ این سؤال را به شما بدهد. با راه‌اندازی اولین اپلیکیشن در ری‌اکت از شروع با پیکربندی صفر، به این اکوسیستم وارد می‌شوید. در طول راه، مقدمه‌ای بر JSX و ReactDOM نیز خواهید یافت. پس برای اولین کامپوننت ری‌اکت خود آماده باشید.

سلام، نام من ری‌اکت است.

چرا باید برای یادگیری ری‌اکت زحمت بکشید؟ در سال‌های اخیر اپ‌های سینگل‌پیج (SPA[4]) محبوب شده‌اند. فریم‌ورک‌هایی مانند Angular، Ember و Backbone به توسعه‌دهندگان جاوااسکریپت برای ساخت وب‌اپلیکیشن‌های مدرن فراتر از استفاده از جاوااسکریپت ساده و جی‌کوئری کمک کرده‌اند. فهرست این راه‌حل‌های محبوب، جامع نیست. طیف گسترده‌ای از فریم‌ورک‌های SPA وجود دارد. زمانی که به تاریخ انتشار آن‌ها توجه می‌کنید می‌بینید که بیش‌تر آن‌ها از نسل اول SPAها هستند: Angular 2010، Backbone 2010، Ember 2011.

ری‌اکت اولین بار توسط فیس‌بوک در سال ۲۰۱۳ انتشار یافت. ری‌اکت نه یک فریم‌ورک SPA، بلکه یک کتاب‌خانه‌ی view است. ری‌اکت V عبارت MVC[5] (کنترلر، ویو، مدل) است، و تنها شما را قادر می‌سازد کامپوننت‌ها را به‌عنوان عناصر قابل‌مشاهده در مرورگر رندر کنید. بااین‌حال، کل اکوسیستم اطراف ری‌اکت امکان ساخت اپ‌های سینگل‌پیج را فراهم می‌سازد.

اما چرا باید از ری‌اکت به‌جای فریم‌ورک‌های نسل اول SPA استفاده کنید؟ درحالی‌که فریم‌ورک‌های نسل اول سعی در حل هم‌زمان بسیاری از موارد داشتند، ری‌اکت تنها به شما کمک می‌کند تا لایه‌ی view خود را ایجاد کنید. ری‌اکت تنها یک کتاب‌خانه است و نه یک فریم‌ورک. ایده‌ی پشت ری‌اکت: ویوی شما یک سلسله‌مراتب از کامپوننت‌های قابل ساخت است.

در ری‌اکت می‌توانید قبل از معرفی جنبه‌های مختلف اپ خود، تمرکز را روی لایه‌ی view حفظ کنید. هرکدام از جنبه‌های دیگر تنها بلوک دیگری برای SPA شماست. این بلوک‌ها برای ساختن اپ‌های کامل ضروری هستند. آن‌ها دو مزیت دارند.

اول، می‌توانید بلوک‌ها را مرحله‌به‌مرحله یاد بگیرید. نیازی نیست نگران درک هم‌زمان همه‌ی آن‌ها باشید. این امر با فریم‌ورک‌ها که تمام بلوک‌ها را از ابتدا در اختیار شما قرار می‌دهند متفاوت است. این کتاب روی ری‌اکت به‌عنوان اولین بلوک تمرکز دارد. بلوک‌های دیگر در ادامه ظاهر می‌شوند.

دوم، تمام بلوک‌ها قابل تعویض با یکدیگر هستند. این مسأله باعث می‌شود اکوسیستم اطراف ری‌اکت به محیطی خلاقانه تبدیل شود. راه‌حل‌های متفاوت با یکدیگر رقابت می‌کنند. شما می‌توانید جذاب‌ترین راه‌حل‌ها را برای خود و کار خودتان انتخاب کنید.

[4] https://en.wikipedia.org/wiki/Single-page_application
[5] https://en.wikipedia.org/wiki/Model–view–controller

نسل اول فریم‌ورک‌های SPA وارد مرحله‌ی سرمایه‌گذاری شده‌اند. آن‌ها سخت و سازمان‌یافته هستند. ری‌اکت خلاق باقی می‌ماند و توسط چندین شرکت پیش‌رو در فناوری مانند «نتفلیکس»، «ایربی‌ان‌بی» و البته «فیسبوک»[6] به کار گرفته می‌شوند. همه‌ی این شرکت‌ها بر آینده‌ی ری‌اکت سرمایه‌گذاری می‌کنند و از ری‌اکت و اکوسیستم اطراف آن راضی هستند.

امروزه احتمالاً ری‌اکت یکی از بهترین گزینه‌ها برای ساختن وب‌اپلیکیشن‌های مدرن است. ری‌اکت تنها لایه‌ی view رو ارائه می‌دهد، اما اکوسیستم ری‌اکت یک چارچوب کاملاً انعطاف‌پذیر و قابل تعویض است[7]. ری‌اکت یک API ظریف، اکوسیستم شگفت‌انگیز، و یک اجتماع عالی دارد. می‌توانید در مورد تجربه‌ی من از «چرا از انگولار به ری‌اکت منتقل شدم[8]» مطالعه کنید. به‌شدت توصیه می‌کنم درکی از دلیل انتخاب ری‌اکت در برابر فریم‌ورک‌ها یا کتاب‌خانه‌های دیگر پیدا کنید. درهرحال همه علاقه‌مندند بدانند ظرف چند سال آینده ری‌اکت ما را کجا خواهد برد.

تمرین:

- مطالعه‌ی مقاله‌ی «چرا از انگولار به ری‌اکت منتقل شدم»[9]
- مطالعه در مورد اکوسیستم انعطاف‌پذیر ری‌اکت[10]
- مطالعه‌ی مقاله‌ی «چگونه یک فریم‌ورک را یاد بگیریم»[11]

6 https://github.com/facebook/react/wiki/Sites-Using-React
7 https://www.robinwieruch.de/essential-react-libraries-framework/
8 https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/
9 https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/
10 https://www.robinwieruch.de/essential-react-libraries-framework/
11 https://www.robinwieruch.de/how-to-learn-framework/

نیازمندی‌ها

اگر از کتاب‌خانه یا فریم‌ورک SPA متفاوتی سراغ ری‌اکت آمده‌اید، باید تابه‌حال با اصول توسعه‌ی وب‌سایت آشنایی داشته باشید. اگر تازه توسعه‌ی وب‌سایت را شروع کرده‌اید باید CSS، HTML و جاوااسکریپت را بلد باشید تا ری‌اکت را یاد بگیرید. این کتاب شمار را به‌آرامی به JavaScript ES6 و فراتر از آن هدایت خواهد کرد. توصیه می‌کنم برای کمک گرفتن یا کمک به دیگران به گروه رسمی Slack[12] این کتاب بپیوندید.

ویرایشگر و ترمینال

پس محیط توسعه چه می‌شود؟ شما به یک ویرایش‌گر یا IDE کارآ و ترمینال (ابزار command line) نیاز دارید. می‌توانید راهنمای تنظیمات[13] من را دنبال کنید. این راهنما برای کاربران Mac تنظیم شده است اما می‌توانید بسیاری از ابزارهای دیگر سیستم‌عامل را جایگزین کنید. ده‌ها مقاله وجود دارد که به شما نشان می‌دهد که چگونه با جزئیات بهتر یک محیط توسعه‌ی وب را برای سیستم‌عامل خود راه‌اندازی کنید.

هم‌چنین می‌توانید از git و GitHub برای خودتان استفاده کنید و در حین انجام تمرینات کتاب پروژه‌های خود و پیش‌رفت آن‌ها را در GitHub برای خود نگه دارید. در مورد چگونگی استفاده از این ابزار، راهنمایی وجود دارد؛ اما استفاده از این راهنما برای خواندن این کتاب اجباری نیست و ممکن است زمانی که بخواهید همه چیز را از ابتدا یاد بگیرید کار بسیار طاقت‌فرسا خواهد شد؛ بنابراین اگر در توسعه‌ی وب تازه‌وارد هستید می‌توانید از این بخش عبور کنید و بر قسمت‌های ضروری کتاب تمرکز کنید.

نود و ان‌پی‌ام

سرانجام به نصب Node و NPM[14] نیاز دارید. هردو این موارد برای مدیریت کتاب‌خانه استفاده می‌شوند و در طول مسیر به آن‌ها احتیاج دارید. در این کتاب external node packageها را از طریق npm (node package manager) نصب خواهید کرد. این node packageها می‌توانند یک کتاب‌خانه یا کل یک فریم‌ورک باشند.

[12] https://slack-the-road-to-learn-react.wieruch.com
[13] https://www.robinwieruch.de/developer-setup/
[14] https://nodejs.org/en/

شما می‌توانید ورژن‌های node و npm خود را در خط فرمان (command line) تأیید کنید. اگر هیچ خروجی‌ای در ترمینال دریافت نکردید ابتدا باید node و npm را نصب کنید. این‌ها فقط ورژن‌های من در زمان نگارش این کتاب هستند:

Command Line:

Node –version
*v8.9.4
Npm –version
*v5.6.0

node و npm

این گفتار آموزش کمی در مورد node و npm به شما می‌دهد. این آموزش کامل نیست، اما تمام ابزارهای لازم را خواهید آموخت. اگر با هردو آن‌ها آشنا هستید می‌توانید از این گفتار عبور کنید.

ابزار (npm) node package manager به شما امکان نصب بسته‌های node خارجی را از خط فرمان می‌دهد. این بسته‌ها می‌توانند مجموعه‌ای از توابع مفید، کتاب‌خانه‌ها یا کل فریم‌ورک‌ها باشند. آن‌ها وابستگی‌های اپلیکیشن شما هستند. می‌توانید این بسته‌ها را یا در فولدر کلی بسته‌ها یا در فولدر محلی پروژه‌ی خود نصب کنید.

بسته‌های کلی نودها همه‌جای ترمینال در دسترس هستند و باید آن‌ها را تنها یک بار در دایرکتوری گلوبال خود نصب کنید. می‌توانید با تایپ کردن این کدها در ترمینال زیر بسته‌ی گلوبال خود را نصب کنید:

Command Line

```
Npm install –g <package>
```

با g– بسته در npm به‌صورت گلوبال نصب می‌شود. بسته‌های محلی در اپ شما مورد استفاده قرار می‌گیرند. برای نمونه، ری‌اکت یک کتاب‌خانه است که به‌عنوان بسته‌ی محلی در اپ شما استفاده می‌شود. می‌توانید آن را از طریق ترمینال با تایپ کد زیر نصب کنید:

Command Line

```
Npm install <package>
```

برای ری‌اکت به این شکل می‌شود:

Command Line

```
Npm install react
```

بسته‌ی نصب‌شده به‌صورت خودکار در یک پوشه به نام /node_modules ظاهر خواهد شد و در فایل package.json در کنار دیگر وابستگی‌های شما لیست خواهد شد.

اما در اولین مرحله چگونه فولدر node_modules/ و package.json را در پروژه‌ی خود آغاز کنیم؟ یک دستور npm برای راه‌اندازی یک پروژه‌ی npm و درنتیجه یک package.json وجود دارد. تنها زمانی که آن فایل را در اختیار دارید می‌توانید بسته‌های محلی جدید را از طریق npm نصب کنید.

Command Line

```
Npm init -y
```

فلگ y– یک میان‌بر برای راه‌اندازی اولیه‌ی تمام پیش‌فرض‌ها در package.json شماست. اگر از این فلگ استفاده نکنید خودتان باید تصمیم بگیرید که چگونه فایل را پیکربندی کنید. پس از راه‌اندازی پروژه‌ی npm می‌توانید بسته‌های جدید را از طریق npm install <package> نصب کنید.

یک نکته‌ی دیگر در مورد package.json. این فایل به شما امکان می‌دهد پروژه‌ی خود را بدون به‌اشتراک گذاشتن تمام بسته‌های node با سایر توسعه‌دهندگان، اشتراک‌گذاری کنید. این فایل دارای تمام مرجع‌های بسته‌های node مورد استفاده در پروژه‌ی شماست. این بسته‌ها وابستگی نامیده می‌شوند. هر کسی می‌تواند پروژه‌ی شما را بدون وابستگی‌ها کپی کند. وابستگی‌ها مرجع‌های package.json شما هستند. کسی که پروژه‌های شما را کپی می‌کند می‌تواند به‌راحتی تمام بسته‌ها را با استفاده از npm در خط فرمان نصب کند. اسکریپت نصب npm، تمام وابستگی‌های لیست‌شده در فایل package.json را می‌گیرد و آن‌ها را در فولدر node_modules/folder نصب می‌کند.

در این‌جا می‌خواهم یک فرمان دیگر npm را نیز آموزش دهم:

Command Line

```
Npm install –save-dev <package>
```

فلگ save-dev– حاکی از آن است که بسته‌ی node تنها در محیط توسعه مورد استفاده قرار می‌گیرد. هنگامی که اپ خود را روی یک سرور قرار می‌دهید، این اپ دیگر در تولید استفاده نخواهد شد. این بسته‌ی node از چه نوعی می‌تواند باشد؟ تصور کنید که می‌خواهید اپ خود را با کمک یک بسته‌ی node آزمایش کنید. ابتدا باید این بسته را از طریق npm نصب کنید، اما لازم است آن را از محیط تولید خود حذف کنید. آزمایش فقط در فرآیند توسعه اتفاق می‌افتد، نه زمانی که اپ شما در تولید در حال اجراست. در آن مرحله دیگر نخواهید خواست که برنامه‌ی خود را بیش‌تر از این آزمایش کنید. اپ

باید از قبل آزمایش شده باشد و برای کاربران به‌خوبی کار کند. این تنها یکی از مواردی‌ست که نیاز دارید از save-dev– استفاده کنید.

در مسیر خود با فرمان‌های npm بیش‌تری مواجه خواهید شد؛ اما موارد ذکرشده تا همین‌جا کافی‌ست.

تمرین:

- یک پروژه را با استفاده از npm راه‌اندازی کنید.
  - یک فولدر جدید با استفاده از mkdir <folder_name> ایجاد کنید.
  - با استفاده از cd <folder_name> وارد پوشه شوید.
  - یکی از دو فرمان npm init –y یا npm init را اجرا کنید.
  - یک بسته‌ی محلی مانند ری‌اکت را با استفاده از npm install eract نصب کنید.
  - نگاهی به فایل package.json و فولدر /node_modules بیاندازید.
  - خودتان راه حذف نصب بسته‌ی node ری‌اکت را پیدا کنید.
- در مورد npm[15] بیش‌تر مطالعه کنید.

[15] https://docs.npmjs.com

نصب و راه‌اندازی

روش‌های متعددی برای شروع کار با برنامه‌ی ری‌اکت وجود دارد.

اولین روش استفاده از CDN است. این کار ممکن است از آن‌چه در ابتدا به نظر می‌آید پیچیده‌تر باشد. CDN یک شبکه‌ی تحویل محتوا[16] ست. چندین شرکت CDN دارند و فایل‌ها را به‌طور عمومی برای مصرف‌کنندگان آن‌ها میزبانی می‌کند. این فایل‌ها می‌توانند کتاب‌خانه‌هایی مثل ری‌اکت باشند، زیرا کتاب‌خانه‌ی مجموعه‌ی ری‌اکت درهرحال تنها از فایل جاوااسکریپت react.js تشکیل شده است. این فایل را می‌توان در جایی میزبانی کرد و سپس در اپ از آن استفاده نمود. چگونه از CDN برای شروع ری‌اکت استفاده کنیم؟ می‌توانید تگ <script> را در HTML خود بنویسید که به آدرس CDN اشاره دارد. برای شروع ری‌اکت نیاز به دو فایل (کتاب‌خانه) نیاز دارید: react و react-dom.

Code Playground

```
<script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></\
script>
<script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.developmen\
t.js"></script>
```

اما زمانی که می‌توانید از طریق npm بسته‌های node مانند ری‌اکت را نصب کنید، چرا باید از CDN استفاده کنید؟

هنگامی که برنامه‌ی شما یک فایل package.json دارد، می‌توانید react و react-dom را از خط فرمان نصب کنید. لازم است در ابتدا با npm init –y فایل package.json را ایجاد کنید. می‌توانید به‌وسیله‌ی npm، بسته‌های چندگانه‌ی node را نصب کنید.

Command Line

```
Npm install react react-dom
```

این رویکرد اغلب برای اضافه کردن ری‌اکت به یک اپ موجود تحت مدیریت npm استفاده می‌شود.

متأسفانه این تمام کار نیست. شما باید برای آگاهی اپ خود از JSX (سینتکس در ری‌اکت) و جاوااسکریپت ES6 از Babel[17] استفاده کنید. «بابل» کدهای شما را ترجمه می‌کند تا مرورگرها بتوانند جاوااسکریپت ES6 و JSX را تفسیر کنند. همه‌ی مرورگرها قادر به تفسیر syntsx نیستند. تنظیمات شامل مقدار زیادی پیکربندی و استفاده از ابزارهاست. مواجهه با تمام این تنظیمات و پیکربندی‌ها می‌تواند برای تازه‌واردان به ری‌اکت غافلگیرکننده باشد.

[16] https://en.wikipedia.org/wiki/Content_delivery_network
[17] https://babeljs.io

به همین دلیل، فیس‌بوک اپ create-react را به‌عنوان پیکربندی صفر ری‌اکت معرفی کرده است. گفتار بعد به شما این امکان را می‌دهد که با استفاده از این ابزار بوت‌استرپینگ، اپ خود را راه‌اندازی کنید.

تمرین:

در مورد react installation[18] بیش‌تر بخوانید.

راه‌اندازی پیکربندی صفر

در مسیر یادگیری ری‌اکت، برای راه‌اندازی اپ خود از create-react-app[19] استفاده خواهید کرد. این یک کیت آغازگر تنظیم-صفر برای راه‌اندازی ری‌اکت است که در سال ۲۰۱۶ توسط فیس‌بوک معرفی شده است. ۹۶% کاربران آن را به مبتدیان توصیه کرده‌اند.[20] در create-react-app ابزار و پیکربندی در پس‌زمینه تکامل می‌یابد، درحالی‌که تمرکز بر روی اجرای برنامه است.

برای شروع باید بسته را به شکل بسته‌ی هانی node نصب کنید. پس از یک بار نصب، همیشه برای راه‌اندازی مجدد ری‌اکت آن را در اختیار خواهید داشت.

Command Line

```
Npm install –g create-react-app
```

می‌توانید برای بررسی موفقیت‌آمیز بودن نصب، ورژن create-react-app را روی خط فرمان خود چک کنید:

Command Line

```
create-react-app –version

*v1.5.1
```

[18] https://reactjs.org/docs/getting-started.html
[19] https://github.com/facebook/create-react-app
[20] https://twitter.com/dan_abramov/status/806985854099062785

حالا می‌توانید اولین برنامه‌ی ری‌اکت خود را بوت‌استرپ کنید. ما آن را hackernews نامیده‌ایم، اما شما می‌توانید نام دیگری برای آن انتخاب کنید. بوت‌استرپ آن چند ثانیه طول می‌کشد. پس از آن به‌راحتی وارد پوشه شوید:

Command Line

```
create-react-app hackernews

cd hackernews
```

حالا می‌توانید برنامه را در ویرایش‌گر خود باز کنید. ساختار پوشه یا تغییر آن بسته به ورژن create-react-app باید به شکل زیر نمایش داده شود:

Folder Structure

```
Hackernews/
        README.md
        Node_modules/
        Package.json
        .gitignore
        Public/
                Favicon.ico
                Index.html
                Manifest.json
        Src/
                App.css
                App.js
                App.test.js
                Index.css
                Index.js
                Index.svg
                Logo.svg
                registerServiceWorker.js
```

یک شمای کلی از فولدر و فایل‌ها. اگر در ابتدا همه‌ی آن‌ها را درک نکنید هیچ اشکالی ندارد.

- README.md: فرمت md. نشان می‌دهد که فایل موردنظر یک فایل markdown است. «مارک‌داون» به‌عنوان یک زبان نشانه‌گذاری سبک با سینتکس فرمت متن ساده مورد استفاده قرار می‌گیرد. بسیاری از پروژه‌های سورس کد از فایل README.md استخراج می‌شوند تا دستورالعمل‌های اولیه در مورد پروژه را به شما بدهند. هنگامی که پروژه‌ی خود را از یک پلتفرم مانند GitHub راه‌اندازی می‌کنید، فایل README.md زمانی که به کدها دسترسی پیدا می‌کنید، محتوای خود را به‌صورت برجسته نمایش می‌دهد. از آن‌جا که پروژه را با —create react-app باز کرده‌اید، README.md شما باید با فایل رسمی create-react-app نمایش‌داده‌شده در ریپازیتوری گیت‌هاب[۲۱] یکسان باشد.
- /Node-modules: این پوشه حاوی تمام بسته‌های node است که توسط npm نصب شده است. از آن‌جا که شما از create-react-app استفاده می‌کنید، چند ماژول node باید از پیش برای شما نصب شده باشد. معمولاً با این پوشه سروکار ندارید، بلکه تنها بسته‌های node را با استفاده از npm در خط فرمان نصب یا حذف می‌کنید.
- Package.json: این فایل لیستی از وابستگی‌های node و دیگر پیکربندی‌های پروژه را نشان می‌دهد.
- gitignore.: این فایل تمام فایل‌ها و پوشه‌هایی را نشان می‌دهد که در زمان استفاده از گیت نباید به گیت ریموت شما اضافه شوند. آن‌ها تنها باید در پروژه‌های محلی شما حضور داشته باشند. فولدر /node-modules یکی از این موارد است. کافی‌ست فایل package.json را، بدون نیاز به اشتراک‌گذاری کل پوشه‌ی وابستگی‌ها، با همتایان خود به اشتراک بگذارید تا بتوانند همه‌ی وابستگی‌ها را در پروژه‌های خود نصب کنند.
- /Public: این پوشه تمام فایل‌های شما را هنگام ساخت پروژه برای تولید در خود نگه می‌دارد. در نهخایت، تمام کدهای نوشته‌شده‌ی شما در زمان ساخت پروژه در پوشه‌ی /src در چند فایل دسته‌بندی می‌شوند و در پوشه‌ی public قرار می‌گیرند.
- Manifest.json و registerServiceWorker.js: در این مرحله در مورد این فایل‌ها نگران نباشید. در این پروژه به آن‌ها نیازی نداریم.

به هر حال نیازی نیست با تمام فایل‌ها و پوشه‌های ذکر شده سروکار داشته باشید. در ابتدا تمام آن‌چه نیاز دارید در پوشه‌ی /src قرار گرفته است. تمرکز اصلی روی فایل src/App.js برای پیاده‌سازی کامپوننت‌های ری‌اکت است. این فایل

[21] https://github.com/facebook/create-react-app

برای اجرای اپ شما مورد استفاده قرار می‌گیرد، اما بعداً ممکن است بخواهید اجزاء خود را به چند فایل تقسیم کنید، درحالی‌که هریک از این فایل‌ها ممکن است یک یا چند کامپوننت در خود داشته باشد.

علاوه بر این، یک فایل src/App.test.js برای آزمایش‌های خود و یک فایل src/index.js به‌عنوان نقطه‌ی ورود به دنیای ری‌اکت در اختیار دارید. در گفتارهای آینده با هردو این فایل‌ها آشنا خواهید شد. به علاوه، یک فایل src/index.css و یک فایل src/App.css برای استایل‌دهی به کامپوننت‌ها وجود دارد. همه‌ی آن‌ها در هنگام باز شدن در حالت پیش‌فرض خود وجود دارند.

برنامه‌ی create-react-app یک پروژه‌ی npm است. شما می‌توانید از npm برای نصب و حذف بسته‌های node بر پروژه‌ی خود استفاده کنید. علاوه بر این، اسکریپت npm با دستور زیر به خط فرمان شما می‌آید:

Command Line:

// Runs the application in http://localhost:3000

Npm start

// Runs the tests

Npm test

// Builds the application for production

Npm run build

اسکریپت‌ها در package.json شما تعریف شده‌اند. اکنون اپ ری‌اکت شما راه‌اندازی شده است. بخش هیجان‌انگیز ماجرا در تمرین‌ها قرار دارد تا در نهایت بوت شدن اپ شما را در مرورگر اجرا کند.

تمرین:

- اپ خود را npm start کنید و اپ خود را در مرورگر ببینید. (با فشار دادن control+c می‌توانید از دستور خارج شوید.)
- اسکریپت npm test تعاملی را اجرا کنید.

- اسکریپت npm run build را اجرا کنید و مطمئن شوید که پوشه‌ی build/ به پروژه اضافه شده است (بعداً می‌توانید آن را حذف کنید؛ توجه داشته باشید که پوشه‌ی build را می‌توان بعداً برای اجرای اپ[۲۲] استفاده کرد).
- با ساختار پروژه آشنا شوید.
- با محتوای فایل‌ها آشنا شوید.
- درباره‌ی اسکریپت‌های npm و create-react-app[۲۳] بیش‌تر بدانید.

[22] https://www.robinwieruch.de/deploy-applications-digital-ocean/
[23] https://github.com/facebook/create-react-app

مقدمه‌ای بر JSX

دراین‌جا می‌توانید JSX را بشناسید. JSX سینتکس ری‌اکت است. چنان که پیش از این ذکر شد، create-react-app برای شما یک اپ ساخته است. تمام فایل‌ها با پیاده‌سازی پیش‌فرض همراه است. این‌جا بهتر است به سورس کدها نگاهی بیاندازیم. تنها فایلی که در ابتدا با آن کار می‌کنید src/App.js است.

src/App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
	render() {
		return (
			<div className="App">
				<header className="App-header">
					<img src={logo} className="App-logo" alt="logo" />
					<h1 className="App-title">Welcome to React</h1>
				</header>
				<p className="App-intro">
					To get started, edit <code>src/App.js</code> and save to reload.
				</p>
			</div>
		);
	}
}
export default App;
```

سعی کنید عبارت‌های import/export و تعیین کلاس‌ها شما را گیج نکند. این ویژگی‌ها همان جاوااسکریپت هستند. آن‌ها را در گفتارهای آینده بررسی خواهیم کرد.

در این فایل، شما یک کامپوننت ری‌اکت کلاس ES6 به نام App دارید. این اعلام کامپوننت است. اساساً بعد از تعریف کامپوننت می‌توانید آن را به عنوان یک المنت در همه‌جای اپ خود استفاده کنید. این کار یک نمونه‌ی آنی از کامپوننت شما را تولید می‌کند، یا به عبارت دیگر: کامپوننت نمونه‌سازی می‌شود.

المنت برگشت‌داده‌شده با متد () render مشخص شده است. المنت‌ها، همان عناصری هستند که کامپونت‌ها از آن‌ها ساخته شده‌اند. دانستن تفاوت میان کامپوننت‌ها، نمونه‌ها و المنت‌ها می‌تواند مفید باشد.

خیلی سریع خواهیم دید که کدام قسمت از اپ نمونه‌سازی شده است. در غیر این صورت خروجی ارائه‌شده در مرورگر را نخواهید دید. کامپوننت App تنها تعریف است و نه کاربرد. باید این کامپوننت را در جایی در فایل JSX با <App /> نمونه‌سازی کنید.

محتوای رندر شده در بلوک رندر بسیار شبیه به HTML خواهد بود، اما در واقع JSX است. JSX شما را قادر می‌سازد تا HTML و جاوااسکریپت را ادغام کنید. زمانی که شما عادت داشته باشید HTML و جاوااسکریپت را جداجدا استفاده کنید، این کار قدرتمند اما گیج‌کننده خواهد بود. به همین دلیل، یک راه خوب برای شروع استفاده از HTML پایه در JSX است. در ابتدا، تمام محتوای اضافی و گیج‌کننده درون فایل را پاک کنید.

src/App.js

```
import React, { Component } from 'react';

import './App.css';

class App extends Component {

render() {

return (

<div className="App">

<h2>Welcome to the Road to learn React</h2>

</div>

);

}

}

export default App;
```

در این‌جا در متد () render فقط HTML را بدون جاوااسکریپت خروجی می‌گیرید. بیایید Welcome to the Road to learn React را به عنوان یک متغیر تعریف کنیم. در JSX متغیر را می‌توان با استفاده از آکولاد ایجاد کرد.

src/App.js

```
import React, { Component } from 'react';

import './App.css';

class App extends Component {

render() {
```

```
var helloWorld = 'Welcome to the Road to learn React';

return (

<div className="App">

<h2>{helloWorld}</h2>

</div>

);

}

}

export default App;
```

زمانی که اپ خود را در خط فرمان دوباره با npm start راه‌اندازی کردید، باید اپ کار کند.

علاوه بر این، ممکن است متوجه ویژگی className شده باشید. این مفهوم، ویژگی کلاس را در HTML نشان می‌دهد. به دلایل فنی، JSX مجبور بود تعدادی از ویژگی‌های داخلی HTML را در خود جایگزین کند. می‌توانید در این لینک تمام ویژگی‌های جایگزین‌شده‌ی HTML در ری‌اکت[۲۴] را بیابید. همه‌ی آن‌ها از قرارداد camelCase پیروی می‌کنند. در مسیر یادگیری ری‌اکت، با برخی از ویژگی‌های خاص JSX آشنا خواهید شد.

تمرین:

- متغیرهای بیش‌تری تعریف کنید و آن‌ها را در JSX خود رندر کنید.
  - از یک شیء پیچیده برای نشان دادن یک کاربر با نام و نام خانوادگی استفاده کنید.
  - ویژگی‌های کاربر را در JSX[۲۵] خود رندر کنید.
- درباره‌ی JSX بیش‌تر بخوانید.
- در مورد کامپونت‌ها، المنت‌ها و نمونه‌های ری‌اکت[۲۶] بیش‌تر بخوانید.

---

[24] https://reactjs.org/docs/dom-elements.html#all-supported-html-attributes
[25] https://reactjs.org/docs/introducing-jsx.html
[26] https://facebook.github.io/react/blog/2015/12/18/react-components-elements-and-instances.html

Const و let در ES6

فکر می‌کنم متوجه شده‌اید که متغیر helloWorld را با عبارت var تعریف کردیم. در جاوااسکریپت ES6 دو گزینه‌ی دیگر برای اعلام متغیرهای شما وجود دارد: const و let. در جاوااسکریپت ES6 به ندرت var را خواهید یافت.

یک متغیر تعریف‌شده با const را نمی‌توان دوباره تعریف یا دوباره مقداردهی کرد. نمی‌توان آن را تکامل (تغییر یا بهبود) داد. با استفاده از آن، ساختارهای تغییرناپذیر را خواهید پذیرفت. هنگامی که ساختار داده تعریف شد، دیگر نمی‌توانید آن را تغییر دهید.

Code Playground

```
// not allowed

const helloWorld = 'Welcome to the Road to learn React';

helloWorld = 'Bye Bye React';
```

متغیر تعریف‌شده توسط let را می‌توان تکامل داد.

Code Playground

```
// allowed

let helloWorld = 'Welcome to the Road to learn React';

helloWorld = 'Bye Bye React';
```

می‌توانید زمانی که نیاز به مقداردهی مجدد دارید از let استفاده کنید.

با این وجود، در مورد const باید مراقب باشید. متغیر تعریف‌شده با const را نمی‌توان تغییر داد؛ اما زمانی که متغیر یک آرایه یا شیء باید مقدار آن می‌تواند به‌روزرسانی شود؛ اما این مقدار قابل تکامل نیست.

Code Playground

```
// allowed

const helloWorld = {

text: 'Welcome to the Road to learn React'

};
```

```
helloWorld.text = 'Bye Bye React';
```

اما چه زمانی باید از هرکدام از آن‌ها استفاده کرد؟ نظرات مختلفی در این باره وجود دارد. من پیشنهاد می‌کنم از const هر زمان که خواستید استفاده کنید. این نشان می‌دهد که شما می‌خواهید ساختار داده‌های خود را تغییرناپذیر کنید، گرچه مقدار متغیر خود را قابل تغییر نگه دارید. اگر می‌خواهید مقادیر متغیر خود را تغییر دهید از let استفاده کنید.

ثبات در ری‌اکت و اکوسیستم آن پذیرفته شده است. به همین دلیل است که زمانی که یک متغیر را تعریف می‌کنید، const باید انتخاب پیش‌فرض شما باشد. با این حال در اشیاء پیچیده می‌توان مقادیر درون آن‌ها را تغییر داد. حواستان به این رفتار باشد.

در اپ، باید const را به‌جای var استفاده کنید.

src/App.js

```
import React, { Component } from 'react';
import './App.css';
class App extends Component {
render() {
const helloWorld = 'Welcome to the Road to learn React';
return (
<div className="App">
<h2>{helloWorld}</h2>
</div>
);
}
}
export default App;
```

تمرین:

- در مورد ES6 sonst بیش‌تر بخوانید.
- در مورد ES6 let بیش‌تر بخوانید.
- در مورد ساختار داده‌های غیرقابل‌تغییر بی‌تر تحقیق کنید.
    - چرا در برنامه‌ریزی معنا پیدا می‌کنند؟
    - چرا آن‌ها در ری‌اکت و اکوسیستم آن استفاده می‌شوند؟

ری‌اکت دام (ReactDOM)

قبل از این که به کار با کامپوننت App را ادامه دهید، ممکن است دوست داشته باشید بدانید از این کامپوننت کجا استفاده شده است. کامپوننت App در نقطه‌ی ورود شما به دنیای ری‌اکت قرار دارد.

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';
ReactDOM.render(
<App />,
document.getElementById('root')
);
```

اساساً ()ReactDOM.render از یک DOM node در HTML شما برای جایگزینی آن با JSX استفاده می‌کند. به همین دلیل شما به راحتی می‌توانید ری‌اکت را با هر اپ خارجی ادغام کنید. استفاده‌ی چندباره از ()ReactDOM.render در اپ ممنوع نیست. شما می‌توانید آن را در بوت‌استرپ سینتکس ساده‌ی JSX، در کامپوننت ری‌اکت، کامپوننت چندگانه‌ی ری‌اکت، یا یک اپ کامل چندین بار استفاده کنید؛ اما در یک اپ ساده‌ی ری‌اکت فقط یک بار برای بوت شدن تمام کامپوننت از آن استفاده می‌کنید.

()React.DOM render دو آرگومان می‌گیرد. اولین آرگومان JSX است که رندر می‌شود. آرگومان دوم همان جایی‌ست که اپ ری‌اکت به HTML شما متصل است. انتظار می‌رود که المنتی با id= ‘root’ حضور داشته باشد. برای پیدا کردن این آی‌دی می‌توانید فایل public/index.html خود را باز کنید.

پیاده‌سازی ()React.DOM render پیش از این در کامپوننت اپ شما در حال اجراست. با این حال، به‌کارگیری یک JSX ساده‌تر هم اشکالی ندارد، تا زمانی که فقط با JSX سروکار داشته باشید. نیازی نیست حتماً شبیه‌سازی یک کامپوننت باشد.

Code Playground

```
ReactDOM.render(
<h1>Hello React World</h1>,
document.getElementById('root')
);
```

تمرین:

- فایل public/index.html را باز منید تا ببینید اپ ری‌اکت به کدام قسمت HTML شما متصل می‌شود.
- در مورد رندر کردن المنت‌ها در ری‌اکت[۲۷] بیش‌تر بخوانید.

[27] https://reactjs.org/docs/rendering-elements.html

Hot Module Replacement

می‌توانید در فایل src/index.js یک عمل انجام دهید تا تجربه‌ی خود را به‌عنوان یک توسعه‌دهنده بهبود ببخشید. البته این کار اختیاری است و نباید در ابتدای یادگیری ری‌اکت بر شما غلبه کند.

در create-react-app یک ویژگی وجود دارد که وقتی کد خود را تغییر می‌دهید، مرورگر به‌طور خودکار صفحه را به‌روزرسانی می‌کند. این ویژگی را با تغییر helloworld در فایل src/app.js امتحان کنید. مرورگر باید صفحه را به‌روزرسانی کند؛ اما راه بهتری برای انجام آن وجود دارد.

Hot Module Replacement یا (HMR) یک ابزار برای بارگیری مجدداپ شما در مرورگر است. مرورگر عمل رفرش کردن صفحه را انجام نمی‌دهد. شما می‌توانید به‌راحتی آن را در create-react-app فعال کنید. در src/index.js، نقطه‌ی ورود شما به ری‌اکت، باید یک پیکربندی کوچک را انجام دهید.

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import './index.css';
ReactDOM.render(
<App />,
document.getElementById('root')
);
if (module.hot) {
module.hot.accept();
}
```

همین. برای تغییر متغیر helloworld در فایل src/app.js دوباره تلاش کنید. مرورگر نباید صفحه را رفرش کند، اما اپ را بارگذاری مجدد می‌کند و خروجی جدید را نمایش می‌دهد. HMR دارای مزایای متعددی‌ست:

تصویر کنید با استفاده از دستورات console.log() در حال دی‌باگ کردن هستید. این عبارات در کنسول باقی خواهند ماند، حتی اگر کدتان را تغییر دهید؛ زیرا مرورگر دیگر صفحه را رفرش نمی‌کند. این اتفاق می‌تواند برای اهداف دی‌باگ کردن مناسب باشد.

در یک اپ در حال رشد، رفرش صفحه باعث پایین آمدن بهره‌وری شما می‌شود. باید صبر کنید تا صفحه بارگیری شود. در یک اپ بزرگ، بارگیری صفحه می‌تواند چندین ثانیه طول بکشد. HMR می‌تواند این اشکال را برطرف کند.

بزرگ‌ترین مزیت آن این است که شما می‌توانید state اپ را با HMR نگه دارید. تصور کنید یک دیالوگ با چندین مرحله در اپ خود دارید و شما در مرحله‌ی ۳ قرار دارید. در واقع HMR یک ویزارد است. بدون آن شما کد را تغییر می‌دهید و مرورگر صفحه را رفرش می‌کند. شما باید مجدداً دیالوگ را باز کنید و از مرحله‌ی ۱ تا مرحله‌ی ۳ حرکت کنید. با HMR دیالوگ شما در مرحله‌ی ۳ باز است. حتی اگر کد تغییر کند، state برناخمه را نگه می‌دارد. HMR اپ شما را مجدداً بارگذاری می‌کند اما صفحه را نه.

تمرین:

- چند بار کد src/app.js را تغییر دهید تا HMR را در عمل ببینید.
- ۱۰ دقیقه‌ی اول «Live React: Hot Reloading with Time Travel[۲۸]» با اجرای «دَن آبراموف» را تماشا کنید.

ادغام جاوااسکریپت در JSX

بهتر است به کامپوننت App برگردیم. تاکنون چند متغیر اولیه را در فایل JSX خودتان رندر کرده‌اید. اکنون باید لیستی از آیتم‌ها را رندر کنید. این لیست در ابتدا شامل داده‌های نمونه ایت، اما در ادامه داده‌ها را از یک API[۲۹] خروجی می‌گیرید. این کار بسیار هیجان‌انگیزتر خواهد بود.

در ابتدا باید فهرستی از آیتم‌ها ایجاد کنید.

src/App.js

```
import React, { Component } from 'react';
```

[28] https://www.youtube.com/watch?v=xsSnOQynTHs
[29] https://www.robinwieruch.de/what-is-an-api-javascript/

```
import './App.css';

const list = [

{

title: 'React',

url: 'https://facebook.github.io/react/',

author: 'Jordan Walke',

num_comments: 3,

points: 4,

objectID: 0,

},

{

title: 'Redux',

url: 'https://github.com/reactjs/redux',

author: 'Dan Abramov, Andrew Clark',

num_comments: 2,

points: 5,

objectID: 1,

},

];

class App extends Component {

...

}
```

این داده‌های نمونه، نماینده‌ی داده‌هایی هستند که ما بعداً از یک API دریافت می‌کنیم. در این لیست، هر آیتم یک عنوان، یک url و یک نویسنده دارد. علاوه بر این، هریک از آن‌ها دارای یک آی.دی، امتیاز (که نشان می‌دهد آن بخش چقدر محبوب است)، و یک عدد هستند که تعداد نظرات را نشان می‌دهد.

حالا می‌توانید از قابلیت map ساخته شده در جاوااسکریپت، در JSX خود استفاده کنید. این کار شما را قادر می‌سازد تا لیست آیتم‌های خود را برای نمایش آماده کنید. باز هم باید برای جاوااسکریپت خود در JSX از آکولاد استفاده کنید.

src/App.js

```
class App extends Component {
render() {
return (
<div className="App">
{list.map(function(item) {
return <div>{item.title}</div>;
})}
</div>
);
}
}
export default App;
```

استفاده از جاوااسکریپت در HTML، در JSX بسیار قدرتمند است. به‌طور معمول ممکن است شما از map برای تبدیل یک لیست از آیتم‌ها به لیست دیگر استفاده کنید. در این‌جا از map برای تبدیل یک لیست از آیتم‌ها به المنت‌های HTML استفاده می‌کنید.

تا این‌جای کار، فقط عنوان هر آیتم نمایش داده خواهد شد. بهتر است این‌جا برخی از ویژگی‌های آیتم را نمایش دهیم.

src/App.js

```
class App extends Component {
render() {
return (
<div className="App">
{list.map(function(item) {
return (
<div>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
</div>
);
})}
</div>
);
}
```

```
}

export default App;
```

شما می‌توانید ببینید چگونه عمل‌گر map به‌سادگی در JSX شما ایجاد شده است. هر ویژگی آیتم در یک تگ نمایش داده می‌شود. علاوه بر این، ویژگی url در تگ <a> به‌عنوان href استفاده شده است.

ری‌اکت همه‌ی کارها را برای شما انجام می‌دهد و تمام آیتم‌ها را نشان می‌دهد؛ اما بید یک کمک‌کننده برای ری‌اکت اضافه کنید تا کارآیی آن را بهبود ببخشید. شما باید یک صفت کلیدی به هر المنت حاضر در لیست اختصاص بدهید. به این ترتیب زمانی که لیست تغییر می‌کند، ری‌اکت قادر به شناسایی آیتم‌ها برای اضافه کردن، تغییر و حذف خواهد بود. برای آیتم‌های لیست نمونه، از یک آی.دی به‌عنوان یک صفت کلیدی استفاده شده است.

src/App.js

```
{list.map(function(item) {

return (

<div key={item.objectID}>

<span>

<a href={item.url}>{item.title}</a>

</span>

<span>{item.author}</span>

<span>{item.num_comments}</span>

<span>{item.points}</span>

</div>

);

})}
```

باید اطمینان حاصل کنید که ویژگی کلیدی، یک شناسه‌ی پایدار است. اشتباه استفاده از index آیتم‌ها در آرایه را مرتکب نشوید. آرایه‌ی index به هیچ وجه پایدار نیست. به‌عنوان مثال وقتی ترتیب لیست تغییر پیدا می‌کند، ری‌اکت نمی‌تواند به‌درستی آیتم‌ها را تشخیص دهد.

src/App.js

```
// don't do this

{list.map(function(item, key) {

return (

<div key={key}>

...

</div>

);

})}
```

اکنون هر دو لیست را نمایش می‌دهید. می‌توانید اپ خود را استارت کنید، مرورگر را باز نموده و هردو لیست نمایش‌داده‌شده را مشاهده کنید.

تمرین:

- در مورد لیست‌ها و کلیدهای ری‌اکت[۳۰] بیش‌تر بخوانید.
- مقاله‌ی «standard built-in array functionalities in JavaScript[۳۱]» را خلاصه کنید.
- از عبارات جاوااسکریپت بیش‌تری در JSX خود استفاده کنید.

[30] https://reactjs.org/docs/lists-and-keys.html
[31] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

ارو فانکشن در ES6

جاوااسکریپت ES6 توابع «ارو» (arrow) را برای اولین بار به کار گرفت. یک تابع arrow کوتاه‌تر از عبارت تابع است.

Code Playground

```
// function expression

function () { ... }

// arrow function expression

() => { ... }
```

اما باید از ویژگی‌های آن آگاه باشید. یکی از این ویژگی‌ها رفتار متفاوت با شیء this است. یک عبارت تابع همیشه شیء this خود را تعریف می‌کند. تابع arrow شیء this را هم‌چنان در متن خود دارد. در زمان استفاده از this در تابع arrow گیج نشوید.

یک حقیقت ارزش‌مند در مورد توابع arrow مربوط به پرانتز است. زمانی که تابع فقط یک آرگومان را دریافت می‌کند می‌توانید پرانتز را حذف کنید، اما زمانی که تابع چند آرگومان دریافت می‌کند باید پرانتز را نگه دارید.

Code Playground

```
// allowed

item => { ... }

// allowed

(item) => { ... }

// not allowed

item, key => { ... }

// allowed

(item, key) => { ... }
```

با این حال بیایید به تابع map نگاهی بیاندازیم. شما می‌توانید آن را با استفاده از یک تابع ارو ES6 خلاصه‌تر بنویسید.

src/App.js

```
{list.map(item => {
return (
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
</div>
);
})}
```

علاوه بر این می‌توانید block body، یعنی آکولادها، را در تابع ارو ES6 حذف کنید. در یک بدنه‌ی خلاصه، یک بازگشت ضمنی اتفاق می‌افتد؛ بنابراین شما می‌توانید عبارت return را حذف کنید. این امر در این کتاب اغلب اتفاق خواهد افتاد؛ بنابراین هنگام استفاده از توابع arrow مطمئن شوید تفاوت بین یک بدنه‌ی بلوک و یک بدنه‌ی مختصر را درک می‌کنید.

src/App.js

```
{list.map(item =>
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
```

```
<span>{item.points}</span>

</div>

)}
```

اکنون JSX شما بسیار مختصر و خواناست. JSX عبارت تابع، آکولادها و عبارت return را حذف کرده است. در عوض یک توسعه‌دهنده می‌تواند تمرکز خود را بر روی جزئیات پیاده‌سازی نگه دارد.

تمرین:

- در مورد تابع ارو ES6[۳۲] بیش‌تر بخوانید.

[32] https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions

کلاس‌ها در ES6

در جاوااسکریپت ES6 کلاس‌ها معرفی شده‌اند. یک کلاس به‌طور معمول در زبان‌های برنامه‌نویسی شیءگرا استفاده می‌شود. جاوااسکریپت در الگوهای برنامه‌ریزی خود بسیار اعطاف‌پذیر بوده و هم‌چنان نیز هست. شما می‌توانید برنامه‌نویسی تابعی و برنامه‌ریزی شیءگرا را در کنار هم برای موارد خاص استفاده کنید.

با وجود آن که ری‌اکت درون خود برنامه‌نویسی تابعی را دارد، برای نمونه با ساختار داده‌های تغییرناپذیر، کلاس‌ها از کامپونت‌ها استفاده می‌کنند. آن‌ها کامپوننت‌های کلاس ES6 نامیده می‌شوند. ری‌اکت بخش‌های خوب هردو نمونه‌ی برنامه‌نویسی را با هم ادغام می‌کند.

بهتر است برای امتحان کردن کلاس جاوااسکریپت ES6، کلاس توسعه‌ی زیر را بدون فکر کردن در مورد کامپوننت بررسی کنیم.

Code Playground

```
class Developer {
constructor(firstname, lastname) {
this.firstname = firstname;
this.lastname = lastname;
}
getName() {
return this.firstname + ' ' + this.lastname;
}
}
```

هر کلاس یک کامپوننت سازنده (Developer) دارد که آن را نمونه‌سازی کند. سازنده می‌تواند آرگومان‌هایی را بگیرد تا آن‌ها را به کلاس اختصاص دهد. علاوه بر این، یک کلاس می‌تواند تابع را تعریف کند. از آن‌جا که کلاس با تابع مرتبط است، کل این فرآیند یک «متد» نامیده می‌شود. اغلب به آن متد کلاس می‌گویند.

کلاس Developer فقط اعلام کلاس است. شما می‌توانید چند نمونه از آن را از طریق فراخوانی آن ایجاد کنید. این کلاس شبیه کلاس کامپوننت ES6 است، که یک بار تعریف می‌شود، اما باید آن را در جایی دیگر برای استفاده به کار ببرید.

بهتر است ببینیم چگونه می‌توان یک کلاس را تعریف کرد و چگونه می‌توان از متد آن استفاده نمود.

Code Playground

```
const robin = new Developer('Robin', 'Wieruch');

console.log(robin.getName());

// output: Robin Wieruch
```

ری‌اکت از کلاس‌های جاوااسکریپت ES6 برای کامپوننت کلاس ES6 استفاده می‌کند. شما قبلاً از یک کامپوننت کلاس ES6 استفاده کرده‌اید.

src/App.js

```
import React, { Component } from 'react';

...

class App extends Component {

render() {

...

}

}
```

کلاس اپ در ادامه‌ی کامپوننت گسترش می‌کند. در ابتدا شما کامپوننت اپ را تعریف می‌کنید، اما این کامپوننت از جای دیگری گسترش می‌یابد. گسترش به چه معناست؟ در برنامه‌نویسی شیءگرا شما باید قاعده‌ی وراثت را به یاد داشته باشید. از این مفهوم برای انتقال ویژگی‌ها از یک کلاس به کلاس دیگر استفاده می‌شود.

عمل‌کرد کلاس اپ از کلاس کامپوننت گسترش می‌یابد. به عبارت مشخص‌تر، کلاس اپ ویژگی‌هایی را از کلاس کامپوننت به ارث می‌برد. از کلاس کامپوننت برای گسترش کلاس پایه‌ی ES6 به کلاس کامپوننت ES6 استفاده می‌شود. این کلاس تمام ویژگی‌هایی که کامپوننت ری‌اکت نیاز دارد را ارائه می‌دهد. متد رندر یکی از ویژگی‌هایی‌ست که قبلاً از آن استفاده کرده‌اید. بعداً در مورد سایر متدهای کلاس کامپوننت بیش‌تر یاد خواهید گرفت.

کلاس کامپوننت تمام جزئیات پیاده‌سازی یک کامپوننت ری‌اکت را در خود دارد. این امر توسعه‌دهندگان را قادر می‌سازد تا از کلاس‌ها به‌عنوان کامپوننت در ری‌اکت استفاده کنند.

متدهایی که کامپوننت ری‌اکت ارائه می‌دهد، یک رابط کاربری عمومی‌ست. یکی از این متدها باید overridden شود، زیرا خروجی یک کامپوننت ری‌اکت را تعریف می‌کند. این امر باید تعریف شود.

اکنون شما مبانی اولیه در مورد کلس‌های جاوااسکریپت ES6 را می‌دانید و می‌دانید چگونه از آن‌ها در ری‌اکت استفاده کنید تا کامپوننت‌ها را گسترش دهید. زمانی که کتاب روش‌های چرخه‌ی زندگی را توصیف می‌کند، در مورد متدهای کامپوننت‌ها بیش‌تر یاد خواهید گرفت.

تمرین:

- در مورد کلاس‌های ES6[۳۳] بیش‌تر بخوانید.

اکنون چگونگی بوت‌استرپ کردن اپ ری‌اکت خود را یاد گرفته‌اید! بیایید گفتارهای گذشته را دوباره‌نویسی کنیم:

- ری‌اکت
  - Create-react-app یک اپ ری‌اکت را بوت‌استرپ می‌کند.
  - برای تعریف خروجی کامپوننت ری‌اکت در متد رندر،JSX جاوااسکریپت و HTML را با هم ادغام می‌کند.
  - کامپوننت‌ها، نمونه‌ها و المنت‌ها سه مورد کاملاً متفاوت هستند.
  - ()ReactDOM.render یک نقطه‌ی ورود برای یک اپ ری‌اکت در جهت اتصال ری‌اکت به DOM است.
  - ویژگی‌های ایجاد شده در جاوااسکریپت را می‌توان در JSX استفاده کرد.
    - Map می‌تواند برای رندر کردن یک لیست از آیتم‌ها به‌عنوان المنت‌های HTML استفاده شود.
- ES6
  - متغیرها می‌توانند با const تعریف شوند و let می‌تواند برای موارد خاص استفاده شود.

[33] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

- استفاده از const نسبت به let در اولویت است.
  - توابع arrow می‌توانند برای مختصر کردن شدن کدهای شما مورد استفاده قرار بگیرند.
  - از کلاس‌ها برای تعریف کامپوننت‌ها در ری‌اکت به‌وسیله‌ی گسترش آن‌ها استفاده می‌شوند.

منطقی آن است که در این نقطه یک توقف داشته باشید. مطالب را یاد بگیرید و آن‌ها را به‌کار ببرید. می‌توانید سورس کدهایی که تا کنون نوشته‌اید را امتحان کنید. می‌توانید این سورس کدها را در ریپوزیتوری رسمی[۳۴] آن پیدا کنید.

[34] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.1

مبانی در ری‌اکت

این گفتار شما را در طول مسیر یادگیری مبانی ری‌اکت هدایت می‌کند. این گفتار، state و تعاملات در کامپوننت‌ها را پوشش می‌دهد، زیرا اجزای استاتیک کمی کسل‌کننده هستند، مگر نه؟ علاوه بر این، شما در مورد راه‌های مختلف برای اعلام کامپوننت‌ها و چگونگی خوانا نگه داشتن کامپوننت‌ها و استفاده مجدد از آن‌ها خواهید آموخت. آماده باشید تا روح زندگی را در کامپوننت‌های خودتان بدمید.

State داخلی کامپوننت

State داخلی کامپوننت که به عنوان State لوکال یا محلی نیز شناخته می‌شود، شما را قادر می‌سازد تا ویژگی‌های ذخیره‌شده در کامپوننت خود را ذخیره، ویرایش یا حذف کنید. کلاس کامپوننت ES6 می‌تواند از یک سازنده (constructor) برای مقداردهی اولیه‌ی State داخلی کامپوننت استفاده کند. سازنده فقط یک بار وقتی که مقداردهی اولیه می‌شود، صدا زده می‌شود.

در این‌جا بهتر است یک سازنده‌ی کلاس را تعریف کنیم.

src/App.js

```
class App extends Component {
        constructor(props) {
                super(props);
        }
        ...
}
```

هنگامی که یک سازنده در کلاس کامپوننت ES6 خود دارید، ()superرا اجباری کنید، زیرا کامپوننت App کلاس زیرمجموعه کامپوننت است. از این رو، کامپوننت گسترش کامپوننت App شما تعریف می‌کند. در مورد کلاس کامپوننت ES6بعداً بیش‌تر یاد می‌گیرید.

شما می‌توانید super(props) را نیز فرا بخوانید. این کار، در صورتی که بخواهید به آن‌ها در سازنده دسترسی پیدا کنید، this.props را در سازنده‌ی شما ایجاد می‌کند. در غیر این صورت، هنگام دسترسی به this.props در سازنده شما، آن‌ها *تعریف‌نشده* خواهند بود. بعداً در مورد props در کامپوننت ری‌اکت بیش‌تر خواهید آموخت.

اکنون، در این مورد، state اولیه در کامپوننت شما باید یک لیست نمونه از آیتم‌ها باشد.

src/App.jc

```
const list = [
```

```
        {
                title: 'React',
                url: 'https://facebook.github.io/react/',
                author: 'Jordan Walke',
                num_comments: 3,
                points: 4,
                objectID: 0,
        },
        ...
];
class App extends Component {
        constructor(props) {
                super(props);
                this.state = {
                        list: list,
                };
        }
        ...
}
```

State، با استفاده از شیء this به کلاس محدود می‌شود؛ بنابراین شما می‌توانید به State محلی در کل کامپوننت خود دسترسی داشته باشید. به عنوان مثال، از این حالت می‌توان در متد ()render استفاده کرد. قبلاً یک لیست استاتیک از

آیتم‌ها را در متد ()render خود جایگذاری کرده‌اید که خارج از کامپوننت شما تعریف شده است. حالا زمان استفاده از لیست از State لوکال در اپلیکیشن است.

src/App.js

```
class App extends Component {
    ...
    render() {
        return (
            <div className="App">
                {this.state.list.map(item =>
                    <div key={item.objectID}>
                        <span>
                            <a href={item.url}>{item.title}</a>
                        </span>
                        <span>{item.author}</span>
                        <span>{item.num_comments}</span>
                        <span>{item.points}</span>
                    </div>
                )}
            </div>
        );
    }
}
```

این لیست اکنون قسمتی از کامپوننت است. این لیست در State داخلی کامپوننت قرار دارد. شما می‌توانید آیتم‌ها را در لیست اضافه کنید، تغییر دهید و یا حذف کنید. هربار که Stateکامپوننت خود را تغییر می‌دهید، متد ()render کامپوننت شما دوباره اجرا می‌شود. به این ترتیب شما می‌توانید به سادگی State داخلی خود را تغییر دهید و اطمینان حاصل کنید که کامپوننت مجدداً اجرا شده و داده‌های صحیح که از State لوکال می‌آید را نمایش می‌دهد.

باید مراقب باشید state را مستقیماً تغییر ندهید. برای تغییر state خود باید از یک متد به نام ()setState استفاده کنید. در گفتار بعد در مورد آن می‌خوانید.

تمرین:

- با state محلی کار کنید
  - Stateهای بیش‌تری در سازنده تعریف کنید.
  - در متد ()render خود از state استفاده کنید و به آن دسترسی پیدا کنید
- درباره‌ی سازنده کلاس‌ES6 [۳۵] بیش‌تر بخوانید

[35] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes#Constructor

مقداردهی اولیه شیء در ES6

در جاوااسکریپت ES6 شما می‌توانید از ویژگی اختصار syntax استفاده کنید تا اشیاء خود را به‌طور مختصرتر ایجاد نمایید.

تصور کنید می‌خواهید شیء زیر را ایجاد کنید:

Code Playground

```
const name = 'Robin';
   const user = {
name: name,
};
```

هنگامی که ویژگی نام در شیء شما، برابر با همان نام متغیر باشد، می‌توانید کدها را به شکل زیر بنویسید:

Code Playground

```
const name = 'Robin';
const user = {
    name,
};
```

در اپلیکیشن، همین کار را می‌توانید انجام دهید. نام متغیر فهرست و نام ویژگی state هم‌نام خواهند بود.

Code Playground

```
// ES5
this.state = {
    list: list,
};
// ES6
this.state = {
    list,
};
```

یکی دیگر از موارد مؤثر کمک‌کننده، استفاده از نام‌های متد به صورت مختصر است. در جاوااسکریپت ES6 شما می‌توانید متدها را در یک کامپوننت به صورت مختصرتر ایجاد کنید.

Code Playground

```
// ES5
var userService = {
getUserName: function (user) {
return user.firstname + ' ' + user.lastname;
},
};
// ES6
const userService = {
getUserName(user) {
return user.firstname + ' ' + user.lastname;
},
};
```

آخرین مورد که البته به اندازه‌ی سایر موارد اهمیت دارد این است که شما همیشه می‌توانید از نام محتوای ویژگی‌های محاسبه‌شده در جاوااسکریپت ES6 استفاده کنید.

Code Playground

```
// ES5
var user = {
name: 'Robin',
};
// ES6
const key = 'name';
const user = {
[key]: 'Robin',
};
```

شاید نام محتوای ویژگی محاسبه‌شده هنوز برای شما معنا ندارد. چرا به آن‌ها نیاز دارید؟ در گفتارهای آینده به یک نقطه می‌رسید که می‌توانید از آن‌ها برای اختصاص دادن مقادیر به key به یک شیوه‌ی داینامیک در شیء استفاده کنید. این کار برای ایجاد کدهای جاوااسکریپت بسیار مناسب است.

تمرین:

- مقداردهی اولیه اشیاء با ES6 را تمرین کنید.
- درباره مقداردهی اشیاء در ES6[36] بیش‌تر بخوانید.

[36] https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Object_initializer

جریان دیتای یک‌طرفه

اکنون می‌توانید در کامپوننت App خود تعدادی state داخلی داشته باشید. با این حال، هنوز state لوکال را دستکاری نکرده‌اید. State استاتیک است و بنابراین یک کامپوننت است. یک راه خوب برای تجربه دستکاری state این است که چند تعامل در کامپوننت داشته باشید.

بهتر است یک دگمه برای هر آیتم در لیست، نمایش دهیم. این دگمه «رد کردن» نام دارد، و قصد دارد آیتمی را از لیست حذف کند. این کار می‌تواند در نهایت زمانی که فقط می‌خواهید فهرستی از آیتم‌های خوانده‌نشده را نگه دارید و آیتم‌هایی که مورد علاقه شما نیست را رد کنید، مفید باشد.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
{this.state.list.map(item =>
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
<span>
<button
onClick={() => this.onDismiss(item.objectID)}
type="button"
>
Dismiss
</button>
</span>
</div>
```

```
)}
</div>
);
}
}
```

متد کلاس ()onDismiss هنوز تعریف نشده است. ما این کار را در یک لحظه انجام خواهیم داد؛ اما در حال حاضر باید تمرکز برروی onClick و المنت دگمه باشد. همان‌طور که می‌بینید متد ()onDismiss در هندلر ()onClick توسط یک تابع دیگر محصور می‌شود. این یک تابع محصورکننده، یک تابع arrow است. به این ترتیب می‌توانید از ویژگی آی‌دی هر یک از آیتم‌ها برای شناسایی آیتم‌هایی که حذف می‌شوند استفاده کنید. یک راه دیگر می‌تواند تعریف تابع خارج از هندلر ()onClick باشد، و فقط تابع تعریف‌شده را به هندلر ()onClick ارسال کنیم. در گفتارهای آینده در مورد هندلرها در المنت‌ها بیش‌تر توضیح خواهیم داد.

آیا به چندخطی بودن کدها برای المنت دگمه دقت کردید؟ توجه داشته باشید که عناصر با چند ویژگی، در برخی نقاط فقط در یک خط در هم مخلوط می‌شوند. به همین دلیل المنت دگمه برای خوانایی بیش‌تر از ویژگی‌های چندخطی و فاصله‌گذاری اول خط استفاده می‌کند. این کار اجباری نیست، بلکه فقط یک توصیه برای استایل کد است که من شدیداً توصیه می‌کنم انجام شود.

اکنون باید عمل‌کرد ()onDismiss را اجرا کنید. این عمل‌کرد آی‌دی را می‌گیرد تا آیتم‌ها برای حذف شدن شناسایی شوند. این تابع محدود به کلاس است و بنابراین تبدیل به یک متد کلاس می‌شود. به همین دلیل شما به آن با ()this.onDismiss دسترسی دارید و نه با ()onDisMiss. شیء this نمونه‌ی کلاس شماست. برای تعریف ()onDisMiss به عنوان متد کلاس، باید آن را در سازنده ادغام کنید. ادغام کردن در یکی از گفتارهای آینده توضیح داده خواهد شد.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
```

```
list,
};
this.onDismiss = this.onDismiss.ادغام(this);
}
render() {
...
}
}
```

در مرحله‌ی بعد باید عمل‌کرد آن، یعنی منطق کسب‌وکار، را در کلاس متد خود تعریف کنید. متدهای کلاس را می‌توان به روش‌های زیر تعریف کرد.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
list,
};
this.onDismiss = this.onDismiss.ادغام(this);
}
onDismiss(id) {
...
}
render() {
...
}
}
```

اکنون شما می‌توانید آن‌چه در داخل متد کلاس اتفاق می‌افتد را تعریف کنید. اساساً شما قصد دارید آیتم‌های شناسایی شده توسط آی‌دی را از لیست حذف، و یک لیست به‌روز شده را در state لوکال خود ذخیره کنید. پس از آن، لیست به‌روز شده در متد رندر دوباره اجرا می‌شود تا آن را نمایش دهد. آیتم‌های حذف شده دیگر نباید نمایش داده شوند.

شما می‌توانید با استفاده از قابلیت فیلتر داخلی تعریف‌شده در جاوااسکریپت، یک آیتم را از لیست حذف کنید. تابع فیلتر، یک تابع را به عنوان ورودی می‌گیرد. این تابع به ارزش هر آیتم در لیست دسترسی دارد، زیرا این تابع در طول لیست تکرار می‌شود. به این ترتیب می‌توانید هر یک از آیتم‌های لیست را بر اساس شرایط فیلتر ارزیابی کنید. اگر ارزیابی یک آیتم درست باشد، آیتم در لیست می‌ماند. در غیر این صورت از لیست حذف خواهد شد. علاوه‌براین خوب است بدانید که تابع یک لیست جدید را باز می‌کند و لیست قدیمی را تغییر نمی‌دهد. این فرایند، قرارداد ری‌اکت مبنی بر داشتن ساختار تغییرناپذیر داده‌ها را پشتیبانی می‌کند.

src/App.js

```
onDismiss(id) {
const updatedList = this.state.list.filter(function isNotId(item) {
return item.objectID !== id;
});
}
```

در مرحله‌ی بعد می‌توانید نتایج تابع را گرفته و آن را به تابع فیلتر انتقال دهید.

src/App.js

```
onDismiss(id) {
function isNotId(item) {
return item.objectID !== id;
}
const updatedList = this.state.list.filter(isNotId);
}
```

علاوه‌براین، می‌توانید با استفاده از تابع arrow جاوااسکریپت ES6 این کد را خلاصه‌تر بنویسید.

src/App.js

```
onDismiss(id) {
const isNotId = item => item.objectID !== id;
const updatedList = this.state.list.filter(isNotId);
}
```

شما حتی می‌توانید این کد را، مانند هندلر onClick در دگمه، در یک خط بنویسید، اما ممکن است خوانایی کمتری پیدا کند.

src/App.js

```
onDismiss(id) {
const updatedList = this.state.list.filter(item => item.objectID !== id);
}
```

لیست اکنون آیتم‌های کلیک‌شده را حذف می‌کند. با این حال state هنوز آپدیت نشده است. در نهایت شما می‌توانید از متد کلاس ()setState برای به‌روزرسانی state داخلی کامپوننت استفاده کنید.

src/App.js

```
onDismiss(id) {
const isNotId = item => item.objectID !== id;
const updatedList = this.state.list.filter(isNotId);
this.setState({ list: updatedList });
}
```

اکنون اپ خود را دوباره اجرا کرده و دگمه‌ی «رد کردن» را امتحان کنید. باید کار کند. آن‌چه اکنون تجربه می‌کنید **جریان داده‌ی یک‌طرفه** در ری‌اکت است. شما در view با ()onClick یک فعالیت را شروع می‌کنید، یک تابع یا متد کلاس state داخلی کامپوننت را تغییر می‌دهد، و متد ()render کامپوننت دوباره اجرا می‌شود تا view را به‌روز رسانی کند.

تمرین

- در مورد state و چرخه‌ی زندگی (life cycle)[۳۷] در ری‌اکت بیش‌تر بخوانید.

[37] https://reactjs.org/docs/state-and-lifecycle.html

اتصال دهنده‌ها

هنگام استفاده از کلاس کامپوننت در ری‌اکت ES6 مهم است درباره اتصال در کلاس‌های جاوااسکریپت هم مطالبی یاد بگیریم. در گفتار قبل، متد کلاس ()onDismiss را در سازنده ادغام کردید.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
list,
};
this.onDismiss = this.onDismiss.ادغام(this);
}
...
}
```

اصلاً چرا این کار را انجام می‌دهید؟ مرحله اتصال لازم است، زیرا متدهای کلاس به‌صورت خودکار this را به نمونه‌ی کلاس متصل نمی‌کنند. بهتر است این امر را با کمک کلاس کامپوننت ES6 زیر نشان دهیم.

Code Playground

```
class ExplainادغامlingsComponent extends Component {
onClickMe() {
console.log(this);
}
render() {
return (
<button
onClick={this.onClickMe}
type="button"
>
Click Me
```

```
</button>
);
}
}
```

رندرهای کامپوننت خوب هستند؛ اما هنگامی که روی دگمه کلیک می‌کنید، در گزارش کنسول شما «شناخته‌نشده» خواهید بود. این یک منبع اصلی اشکالات در هنگام استفاده از ری‌اکت است؛ زیرا اگر بخواهید به this.state در متد کلاس خود دسترسی داشته باشید، نمی‌توانید آن را بازیابی کنید زیرا شناخته‌نشده است؛ بنابراین برای این که این دسترسی را در متدهای کلاس خود داشته باشید، باید متدهای کلاس را در this ادغام کنید.

در کلاس کامپوننت زیر، متد کلاس به درستی در سازنده متصل شده است.

Code Playground

```
class ExplainblendingsComponent extends Component {
constructor() {
super();
this.onClickMe = this.onClickMe.ادغام(this);
}
onClickMe() {
console.log(this);
}
render() {
return (
<button
onClick={this.onClickMe}
type="button"
>
Click Me
</button>
);
```

```
}
}
```

هنگام امتحان دوباره‌ی دگمه، شیء this، به‌طور مشخص‌تر نمونه‌ی کلاس، باید تعریف شود و شما اکنون می‌توانید به this.state، یا همانطور که بعدا یاد خواهید گرفت به this.props، دسترسی داشته باشید.

اتصال متد کلاس می‌تواند در جاهایی دیگر نیز رخ دهد. به عنوان مثال، این فرایند می‌تواند در متد کلاس ()render ایجاد شود.

Code Playground

```
class ExplainblendingsComponent extends Component {
onClickMe() {
console.log(this);
}
render() {
return (
<button
onClick={this.onClickMe.ادغام(this)}
type="button"
>
Click Me
</button>
);
}
}
```

اما شما باید از این کار اجتناب کنید زیرا هربار که متد ()render اجرا می‌شود، متد کلاس ادغام می‌شود. اساساً هربار که اپلیکیشن شما آپدیت می‌شود، متد کلاس اجرا می‌شود که منجر به کاربردهای عملی می‌گردد. هنگام ادغام متدهای کلاس در سازنده، شما آن را تنها یک بار در ابتدا هنگامی که کامپوننت تعریف شده است ادغام می‌کنید. این یک رویکرد بهتر برای انجام این کار است.

چیز دیگری که گاهی اوقات مردم با آن مواجه می‌شوند تعریف منطق کسب‌وکار در متدهای کلاس در سازنده است.

Code Playground

```
class ExplainblendingsComponent extends Component {
constructor() {
super();
this.onClickMe = () => {
console.log(this);
}
}
render() {
return (
<button
onClick={this.onClickMe}
type="button"
>
Click Me
</button>
);
}
}
```

شما باید از این کار نیز اجتناب کنید؛ زیرا در طول زمان، سازنده‌ی شما را دچار بی‌نظمی می‌کند. سازنده فقط برای ترسیم کلاس با تمام ویژگی‌های آن است. به همین دلیل است که منطق کسب‌وکار متدهای کلاس باید خارج از سازنده تعریف شود.

Code Playground

```
class ExplainblendingsComponent extends Component {
constructor() {
super();
```

```
this.doSomething = this.doSomething. blend(this);
this.doSomethingElse = this.doSomethingElse.blend(this);
}
doSomething() {
// do something
}
doSomethingElse() {
// do something else
}
...
}
```

در پایان لازم به ذکر است که متدهای کلاس را می‌توان به‌طور خودکار، بدون نیاز به ادغام کردن آن‌ها به‌طور صریح، با استفاده از تابع arrow جاوااسکریپت ES6، ادغام کرد.

Code Playground

```
class ExplainBindingsComponent extends Component {
onClickMe = () => {
console.log(this);
}
render() {
return (
<button
onClick={this.onClickMe}
type="button"
>
Click Me
</button>
);
```

```
}
}
```

اگر اتصالات تکراری در سازنده شما را آزار می‌دهد، می‌توانید با این رویکرد پیش بروید. مستندات رسمی ری‌اکت به ادغام کردن متدهای کلاس در سازنده اشاره دارد. به همین دلیل این کتاب نیز به این روال پیش خواهد رفت.

تمرین

- رویکردهای مختلف اتصال و console log شیء this را تمرین کنید.

هندلر ایونت

این گفتار باید درک عمیقی از هندلر event در المنت‌ها به شما بدهد. شما در اپلیکیشن‌تان از المنت دگمه برای حذف یک آیتم از لیست استفاده می‌کنید.

src/App.js

```
...
<button
onClick={() => this.onDismiss(item.objectID)}
type="button"
>
Dismiss
</button>
...
```

این فرایند از ابتدا یک مورد استفاده‌ی پیچیده بوده است، زیرا شما باید یک ارزش را به متد کلاس انتقال دهید و بنابراین مجبورید آن را در یک تابع (arrow) دیگر بگذارید؛ بنابراین این تابع اساساً باید یک تابع باشد که هندلر event به آن انتقال داده شود. کد زیر کار نمی‌کند، زیرا متد کلاس هنگامی که اپلیکیشن را در مرورگر باز می‌کنید، بلافاصله اجرا می‌شود.

src/App.js

```
...
<button
onClick={this.onDismiss(item.objectID)}
type="button"
>
Dismiss
```

```
</button>

...
```

هنگام استفاده از {onClick() = { do Something تابع ()do Something بلافاصله پس از باز کردن اپلیکیشن در مرورگر شما اجرا می‌شود. عبارت درون هندلر ارزیابی می‌شود. از آن‌جا که مقدار برگشتی تابع دیگر یک تابع محسوب نمی‌شود، وقتی روی دگمه کلیک می‌کنید هیچ اتفاقی نمی‌افتد؛ اما وقتی از {onClick = { do Something() استفاده می‌کنید، چون do Something یک تابع است، هنگام کلیک کردن بر روی دگمه می‌توان آن را اجرا کرد. همان قوانینی برای متد کلاس ()onDismiss اعمال می‌شود که در اپلیکیشن شما استفاده می‌شود.

با این حال استفاده از {onClick = { this.onDismiss کافی نیست، زیرا به هر طریق، لازم است که ویژگی item.objectID به متد کلاس انتقال داده شود تا آیتم‌هایی که قرار است حذف شوند شناسایی گردند. به همین دلیل است که می‌توان آن را به یک تابع دیگر متصل کرد تا به ویژگی‌ها دست پیدا کند. این مفهوم، توابع مرتبه‌ی بالاتر در جاوااسکریپت نامیده می‌شود و به‌طور خلاصه بعداً توضیح داده خواهد شد.

src/App.js

```
...
<button
onClick={() => this.onDismiss(item.objectID)}
type="button"
>
Dismiss
</button>
...
```

یک راه‌حل می‌تواند تعریف تابع در جایی بیرون از کامپوننت باشد و فقط تابع تعریف‌شده را به هندلر انتقال داد. از آن‌جا که هندلر نیاز به دسترسی به یک آیتم منحصربه‌فرد دارد، باید در درون بلوک تابع map وجود داشته باشد.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
{this.state.list.map(item => {
const onHandleDismiss = () =>
this.onDismiss(item.objectID);
return (
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
<span>
<button
onClick={onHandleDismiss}
type="button"
>
Dismiss
</button>
</span>
</div>
```

```
);
}
)}
</div>
);
}
}
```

به هر حال، باید یک تابع وجود داشته باشد که به هندلر المنت انتقال داده شود. به عنوان مثال این کد را امتحان کنید:

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
{this.state.list.map(item =>
...
<span>
<button
onClick={console.log(item.objectID)}
type="button"
>
Dismiss
</button>
```

```
</span>
</div>
)}
</div>
);
}
}
```

این کد زمانی اجرا می‌شود که اپلیکیشن را در مرورگر باز کنید، اما نه زمانی که روی دگمه کلیک کنید. این در حالی‌ست که کد زیر تنها زمانی اجرا می‌شود که روی دگمه کلیک کنید. این یک تابع است که شروع به کار هندلر اجرا می‌شود.

src/App.js

```
...
<button
onClick={function () {
console.log(item.objectID)
}}
type="button"
>
Dismiss
</button>
...
```

برای کوتاه نگه داشتن کد، می‌توانید دوباره آن را به یک تابع arrow جاوااسکریپت ES6 تبدیل کنید. این همان کاری است که ما با متد کلاس ()onDismiss هم انجام دادیم.

src/App.js

```
...
<button
onClick={() => console.log(item.objectID)}
type="button"
>
Dismiss
</button>
...
```

تازه‌واردان ری‌اکت اغلب با موضوع استفاده از تابع در هندلر event مشکل دارند. به همین دلیل است که سعی کردم جزئیات بیش‌تری را در این‌جا توضیح دهم. در پایان، شما باید کد زیر را در دگمه‌ی خود داشته باشید تا یک تابع Arrow جاوااسکریپت ES6 مختصر را در یک خط قرار دهید، که به ویژگی objectID در شیء item دسترسی دارد.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
{this.state.list.map(item =>
<div key={item.objectID}>
...
<span>
<button
onClick={() => this.onDismiss(item.objectID)}
type="button"
>
Dismiss
</button>
```

```
</span>
</div>
)}
</div>
);
}
}
```

یکی دیگر از موضوعات مرتبط با اجرا، که اغلب در مورد آن صحبت می‌شود، نتایج عملی استفاده از تابع arrow در هندلر event است. به عنوان مثال، هندلر onClick برای متد ()onDismiss، این متد را در یک تابع arrow دیگر قرار می‌دهد تا بتواند آی‌دی آیتم را انتقال دهد؛ بنابراین هرزمان که متد ()render اجرا می‌شود، هندلر تابع arrow مرتبه‌ی بالاتر را تعریف می‌کند. این اتفاق می‌تواند بر عملکرد اپلیکیشن شما تأثیر بگذارد اما در بیش‌تر موارد متوجه آن نمی‌شوید. تصور کنید که یک جدول بزرگ داده با ۱۰۰۰ آیتم داشته باشید و هر ردیف یا ستون دارای چندین تابع arrow در یک هندلر event است. پس خیلی مهم است در مورد نتایج اجرا فکر کنید و در نتیجه می‌توانید یک کامپوننت اختصاصی دگمه برای ادغام کردن متد در سازنده پیاده‌سازی کنید. اما قبل از این که این اتفاق بیافتد، یادگیری آن زودهنگام است و بهتر است تمرکز خود را برای یادگیری ری‌اکت نگه دارید.

تمرین

- روش‌های مختلف استفاده از توابع را در هندلر onClick() دگمه‌ی خود تمرین کنید.

تعامل با فرم‌ها و event ها

اکنون بیایید در اپلیکیشن برای تجربه فرم‌ها و event ها در ری‌اکت تعامل دیگری را اضافه کنیم. این تعامل، قابلیت جست‌وجو است. ورودی فیلد جست‌وجو باید مورد استفاده قرار گیرد تا به‌طور موقت لیست خود را بر اساس ویژگی عنوان آیتم، فیلتر کنید. در اولین قدم یک فرم را با یک فیلد ورودی در JSX خود تعریف می‌کنید.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
<form>
<input type="text" />
</form>
{this.state.list.map(item =>
...
)}
</div>
);
}
}
```

در سناریوی زیر، شما در فیلد ورودی تایپ خواهید کرد و لیست را به‌طور موقت توسط عبارت جست‌وجو که در فیلد ورودی آمده است، فیلتر می‌کنید. برای این که بتوانید لیست را براساس فیلد ورودی فیلتر کنید، باید مقدار فیلد ورودی را در state لوکال خود ذخیره کنید؛ اما چطور به مقدار آن دست پیدا می‌کنید؟ شما می‌توانید از **ایونت‌های تولیدی** در ری‌اکت برای دسترسی به این مقدار استفاده کنید.

اجازه دهید یک هندلر onChange برای فیلد ورودی تعریف کنیم.

src/App.js

```
class App extends Component {
...
render() {
```

```
return (
<div className="App">
<form>
<input
type="text"
onChange={this.onSearchChange}
/>
</form>
...
</div>
);
}
}
```

این تابع به کامپوننت وصل است و در نتیجه باید یک متد کلاس را ادغام و آن متد را تعریف کنید.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
list,
};
this.onSearchChange = this.onSearchChange.bind(this);
this.onDismiss = this.onDismiss.bind(this);
}
onSearchChange() {
...
}

...
}
```

هنگام استفاده از یک هندلر در المنت، به event تولیدی ری‌اکت در تابع callback دسترسی خواهید داشت.

src/App.js

```
class App extends Component {
...
onSearchChange(event) {
...
}
...
}
```

این event حاوی مقدار فیلد ورودی در شیء هدف است. از این رو شما با استفاده از this.setState() می‌توانید با عبارت جست‌وجو state لوکال را به‌روز کنید.

src/App.js

```
class App extends Component {
...
onSearchChange(event) {
this.setState({ searchTerm: event.target.value });
}
...
}
```

علاوه‌براین، نباید فراموش کنید که state اولیه برای ویژگی searchTerm در سازنده تعریف شود. فیلد ورودی باید در ابتدا خالی باشد و در نتیجه مقدار هم باید یک رشته خالی باشد.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
list,
searchTerm: '',
};
```

```
this.onSearchChange = this.onSearchChange.bind(this);
this.onDismiss = this.onDismiss.bind(this);
}
...
}
```

اکنون هربار که مقدار فیلد ورودی تغییر کند، مقدار ورودی در state داخلی کامپوننت ذخیره می‌شود.

یک توضیح مختصر در مورد به‌روزرسانی State لوکال در کامپوننت ری‌اکت: فرض بر این است که زمانی که SearchTerm به‌وسیله‌ی this.setState() به‌روزرسانی می‌شود، لیست نیاز دارد آن را برای ذخیره انتقال دهد؛ اما این چنین نیست. This.setState() در ری‌اکت یک ترکیب سطحی است. این ویژگی‌ها را هنگام به‌روزرسانی فقط در state شیء نگه می‌دارد؛ بنابراین state لیست حتی اگر قبلاً یک آیتم از آن را حذف کرده باشید، هنگام به‌روزرسانی ویژگی searchTerm یکسان باقی می‌ماند.

حال بهتر است به اپ خود برگردیم. لیست هنوز براساس مقدار فیلد ورودی ذخیره‌شده در state لوکال، فیلتر نشده است. اساساً باید این لیست را به‌طور موقت بر اساس searchTerm فیلتر کنید. شما تمام موارد لازم برای فیلتر کردن آن را در اختیار دارید. حال چگونه می‌توانیم آن را به‌طور موقت فیلتر کنیم؟ در متد render() شما، قبل از اینکه map را روی لیست قرار دهید می‌توانید یک فیلتر روی آن اعمال کنید. فیلتر فقط تطابق searchTerm با ویژگی عنوان آیتم را ارزیابی می‌کند. شما پیش از این از عملکرد فیلتر ایجاد شده در جاوااسکریپت استفاده کرده‌اید، پس اجازه دهید دوباره آن را انجام دهیم. شما می‌توانید قبل از تابع map، حرکتی گذرا در تابع فیلتر داشته باشید، زیرا تابع فیلتر یک آرایه‌ی جدید را برمی‌گرداند و بنابراین تابع map می‌تواند به‌عنوان یک راه مناسب استفاده شود.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
<form>
<input
type="text"
```

```
onChange={this.onSearchChange}
/>
</form>
{this.state.list.filter(...).map(item =>
...
)}
</div>
);
}
}
```

اجازه دهید اکنون تابع فیلتر را به نحوی متفاوت انجام دهیم. می‌خواهیم آرگومان فیلتر را تعریف کنیم، تابعی که به تابع فیلتر خارج از کلاس کامپوننت ES6 انتقال داده می‌شود. در آن‌جا به state کامپوننت دسترسی نداریم و بنابراین به ویژگی searchTerm برای ارزیابی شرط فیلتر هم دسترسی نداریم. ما باید searchTerm را به تابع فیلتر انتقال دهیم و برای ارزیابی شرایط باید یک تابع جدید را return کنیم. این یک تابع مرتبه‌ی بالاتر است.

من معمولاً توابع مرتبه‌ی بالاتر را ذکر نمی‌کنم اما در یک کتاب ری‌اکت، این کار کاملاً منطقی خواهد بود. منطقی‌ست که در مورد توابع مرتبه‌ی بالاتر اطلاعات داشته باشیم، زیرا ری‌اکت با مفهوم کامپوننت مرتبه‌ی بالاتر سروکار دارد.

در ادامه‌ی این کتاب با این مفهوم آشنا خواهید شد. اکنون اجازه دهید دوباره برروی عملکرد فیلتر تمرکز کنیم. اول، شما باید تابع مرتبه‌ی بالاتر را خارج از کامپوننت App خود تعریف کنید.

src/App.js

```
function isSearched(searchTerm) {
return function(item) {
// some condition which returns true or false
}
}
class App extends Component {
...
}
```

تابع searchTerm را می‌گیرد و یک تابع دیگر را بازمی‌گرداند، زیرا به هر حال عملکرد فیلتر، تابع آن را به عنوان ورودی می‌گیرد. تابع بازگشتی به شیء آیتم دسترسی دارد، زیرا این است که به تابع فیلتر انتقال داده می‌شود. علاوه براین، تابع بازگشتی برای فیلتر کردن لیست بر اساس شرایط تعریف شده در تابع استفاده می‌شود. بیایید شرایط را تعریف کنیم.

src/App.js

```
function isSearched(searchTerm) {
return function(item) {
return item.title.toLowerCase().includes(searchTerm.toLowerCase());
}
}
class App extends Component {
...
}
```

شرط می‌گوید الگوی searchTerm ورودی را با ویژگی عنوان آیتم در لیست شما مطابقت می‌دهید. شما می‌توانید این کار را با قابلیت‌های ساخته‌شده در جاوااسکریپت انجام دهید. فقط زمانی که الگو مطابقت دارد، true را برمی‌گردانید و آیتم در لیست باقی می‌ماند. هنگامی که الگو مطابقت ندارد، آیتم از لیست حذف می‌شود؛ اما در مورد تطبیق الگو دقت کنید: نباید فراموش کنید هر دو رشته با حروف کوچک باشد. در غیر این صورت بین یک عبارت‌های جست‌وجو با عناوین "redux" و "Redux" عدم تطابق وجود خواهد داشت. از آن‌جا که ما درحال کار برروی یک لیست تغییرناپذیر هستیم و با استفاده از تابع فیلتر یک لیست جدید را باز می‌کنیم، لیست اصلی در state لوکال به هیچچ عنوان تغییری نخواهد کرد.

یک مورد دیگر هم قابل ذکر است: ما کمی با استفاده کردن از ویژگی‌های ساخته شده در جاوااسکریپت تقلب کردیم. این ویژگی‌ها متعلق به ES6 هستند. آن‌ها در جاوااسکریپت ES5 چگونه به نظر می‌رسند؟ شما از تابع ()indexof برای گرفتن فهرست آیتم در لیست استفاده می‌کنید. هنگامی که آیتم در لیست حضور دارد، ()Indexof فهرست را در آرایه نشان می‌دهد.

Code Playground

```
// ES5
string.indexOf(pattern) !== -1
// ES6
string.includes(pattern)
```

یکی دیگر از بازسازی‌های بدون ایراد را می‌توان با تابع arrow در ES6 دوباره بازنویسی کرد. این کار تابع را مختصرتر می‌کند:

Code Playground

```
// ES5
function isSearched(searchTerm) {
return function(item) {
return item.title.toLowerCase().indexOf(searchTerm.toLowerCase()) !== -1;
}
}
// ES6
const isSearched = searchTerm => item =>
item.title.toLowerCase().includes(searchTerm.toLowerCase());
```

می‌توان در مورد این که کدام تابع خواناتر است بحث کرد. من شخصاً دومی را ترجیح می‌دهم. اکوسیستم ری‌اکت از بسیاری از مفاهیم برنامه‌نویسی کاربردی استفاده می‌کند. اغلب اتفاق می‌افتد که شما از یک تابع استفاده می‌کنید که یک تابع دیگر را برمی‌گرداند (توابع مرتبه‌ی بالاتر). در جاوااسکریپت ES6، می‌توانید این توابع را با توابع arrow خلاصه‌تر بیان کنید.

آخرین مورد این است که شما باید از تابع تعریف‌شده‌ی isSearched() برای فیلتر کردن لیست خودتان استفاده کنید. شما ویژگی searchTerm را از state لوکال خود انتقال می‌دهید، این ویژگی تابع ورودی فیلتر را بازمی‌گرداند، و بر اساس شرایط فیلتر لیست شما را فیلتر می‌کند. سپس این ویژگی لیست فیلتر شده را برای نمایش هر آیتم موجود در لیست map می‌کند.

src/App.js

```
class App extends Component {
...
render() {
return (
<div className="App">
```

```
<form>
<input
type="text"
onChange={this.onSearchChange}
/>
</form>
{this.state.list.filter(isSearched(this.state.searchTerm)).map(item =>
...
)}
</div>
);
}
}
```

اکنون قابلیت جست‌وجو باید کار کند. این قابلیت را در مرورگر خود امتحان کنید.

تمرین

- در مورد event های ری‌اکت[۳۸] بیش‌تر بخوانید.
- در مورد توابع مرتبه‌ی بالاتر[۳۹] بیش‌تر بخوانید.

[38] https://reactjs.org/docs/handling-events.html
[39] https://en.wikipedia.org/wiki/Higher-order_function

تحول ES6

برای دسترسی آسان‌تر به ویژگی‌ها در اشیاء و آرایه‌ها، یک راه در جاوااسکریپت ES6 وجود دارد. این راه destructuring (تحول) نامیده می‌شود. کد زیر را در جاوااسکریپت ES6 و ES5 مقایسه کنید.

Code Playground

```
const user = {
firstname: 'Robin',
lastname: 'Wieruch',
};
// ES5
var firstname = user.firstname;
var lastname = user.lastname;
console.log(firstname + ' ' + lastname);
// output: Robin Wieruch
// ES6
const { firstname, lastname } = user;
console.log(firstname + ' ' + lastname);
// output: Robin Wieruch
```

درحالی‌که در جاوااسکریپت ES5، هربار که می‌خواهید به یک ویژگی شیء دسترسی داشته باشید باید یک خط بیش‌تر بنویسید، در جاوااسکریپت ES6 می‌توانید همین کار را در یک خط انجام دهید. هنگامی که یک شیء چند ویژگی دارد، بهترین شیوه برای خوانایی بیش‌تر این است که از چندین خط استفاده کنید.

Code Playground

```
const {
firstname,
lastname
} = user;
```

این برای آرایه‌ها نیز صدق می‌کند. شما می‌توانید آن‌ها را نیز destructure کنید. تکرار می‌کنم، چند خط کردن باعث خوانایی بهتر کدهای شما می‌شود.

Code Playground

```
const users = ['Robin', 'Andrew', 'Dan'];
const [
userOne,
userTwo,
userThree
] = users;
console.log(userOne, userTwo, userThree);
// output: Robin Andrew Dan
```

شاید متوجه شده باشید که شیء state لوکال در کامپوننت App می‌تواند به همین روش destructure شود. می‌توانید کدهای فیلتر و map را خلاصه‌تر کنید.

src/App.js

```
render() {
const { searchTerm, list } = this.state;
return (
<div className="App">
...
{list.filter(isSearched(searchTerm)).map(item =>
...
)}
</div>
);
```

شما می‌توانید این کار را به روش ES5 یا ES6 ایجاد کنید:

Code Playground

```
// ES5
var searchTerm = this.state.searchTerm;
var list = this.state.list;
```

```
// ES6
const { searchTerm, list } = this.state;
```

اما از آنجا که این کتاب اکثراً از جاوااسکریپت ES6 استفاده می‌کند بهتر است همین روش را در پیش بگیرید.

تمرین

در مورد destructure در ES6[40] بیش‌تر بخوانید.

[40] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

کامپوننت‌های کنترل‌شده

پیش از این در مورد جریان داده‌ی یک‌طرفه در ری‌اکت یاد گرفته‌اید. همان قانون برای فیلد ورودی هم اعمال می‌شود که state لوکال را با searchTerm برای فیلتر کردن لیست به‌روز می‌کند. هنگامی که state تغییر می‌کند، متد ()render دوباره اجرا می‌شود و برای اعمال شرایط فیلتر، searchTerm جدید از state لوکال را مورد استفاده قرار می‌دهد.

اما آیا چیزی را در المنت ورودی فراموش نکرده‌ایم؟ تگ HTML ورودی به‌همراه یک صفت مقدار استفاده می‌شود. صفت مقدار معمولاً دارای مقداری است که در فیلد ورودی نمایش داده شده است. در این مورد، این ویژگی searchTerm خواهد بود. با این حال به نظر می‌رسد که ما در ری‌اکت به آن نیازی نداریم.

اما این تفکر اشتباه است. عناصر فرم مانند تگ‌های <select>، <textarea>، <input>، state خودشان را در HTML نگه می‌دارند. آن‌ها مقادیر داخلی را هربار که کسی آن را از خارج تغییر می‌دهد، اصلاح می‌کنند. این مفهوم در ری‌اکت، کامپوننت کنترل‌نشده نامیده می‌شود زیرا state خودش را مدیریت می‌کند. در ری‌اکت باید مطمئن شوید که این المنت‌ها را به صورت کامپوننت کنترل‌شده ایجاد کنید.

چگونه باید این کار را انجام دهید؟ شما فقط باید یک المنت مقدار برای فیلد ورودی تعریف کنید. مقدار از قبل در ویژگی searchTerm در ویژگی state ذخیره شده است. بنابراین از همان‌جا به آن دسترسی پیدا کنید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, list } = this.state;
return (
<div className="App">
<form>
<input
type="text"
value={searchTerm}
onChange={this.onSearchChange}
/>
</form>
...
</div>
);
```

```
}
}
```

کار انجام شد. اکنون حلقه‌ی جریان داده‌ی یک‌طرفه برای فیلد ورودی موجود است. State داخلی کامپوننت تنها منبع درست برای فیلد ورودی است.

مدیریت کل state داخلی و جریان داده‌ی یک‌طرفه ممکن است برای شما جدید باشد؛ اما هنگامی که به آن عادت کردید، به جریان طبیعی شما برای پیاده‌سازی فرایندها در ری‌اکت تبدیل خواهد شد. به‌طورکلی، ری‌اکت از طریق جریان داده‌ی یک‌طرفه، الگوی جدیدی را به جهان اپلیکیشن‌های سینگل‌پیج معرفی کرده است. این الگو تاکنون توسط چندین فریم‌ورک و کتابخانه به‌کار گرفته شده است.

تمرین

- در مورد فرم‌ها در ری‌اکت[41] بیش‌تر بخوانید.
- در مورد کامپوننت‌های کنترل‌شده‌ی مختلف[42] بیش‌تر بخوانید.

[41] https://reactjs.org/docs/forms.html
[42] https://github.com/the-road-to-learn-react/react-controlled-components-examples

تقسیم کامپوننت

اکنون شما یک کامپوننت App بزرگ دارید. این کامپوننت به رشد خود ادامه می‌دهد و در نهایت می‌تواند گیج‌کننده شود. می‌توانید به تقسیم آن به قطعات کوچک‌تر اقدام کنید.

بهتر است استفاده از یک کامپوننت برای ورودی جست‌وجو و یک کامپوننت برای لیستی از آیتم‌ها را آغاز کنیم.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, list } = this.state;
return (
<div className="App">
<Search />
<Table />
</div>
);
}
}
```

شما می‌توانید ویژگی‌های کامپوننت‌هایی که می‌توانند از خودشان استفاده کنند را انتقال دهید. در مورد کامپوننت App، لازم است که ویژگی‌های مدیریت در state لوکال و متد کلاس را انتقال دهید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, list } = this.state;
return (
<div className="App">
<Search
value={searchTerm}
onChange={this.onSearchChange}
/>
<Table
```

```
list={list}
Basics in React 63
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
</div>
);
}
}
```

اکنون می‌توانید کامپوننت بعدی را در کامپوننت App خود تعریف کنید. این کامپونت‌ها همچنین کامپونت کلاس ES6 خواهند بود. آن‌ها همانند قبل، المنت‌ها را رندر می‌کنند.

اولین آن، کامپوننت Search است.

src/App.js

```
class App extends Component {
...
}
class Search extends Component {
render() {
const { value, onChange } = this.props;
return (
<form>
<input
type="text"
value={value}
onChange={onChange}
/>
</form>
);
}
}
```

دومین آن کامپوننت Table است.

src/App.js

```
...
class Table extends Component {
render() {
const { list, pattern, onDismiss } = this.props;
return (
<div>
{list.filter(isSearched(pattern)).map(item =>
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
<span>
<button
onClick={() => onDismiss(item.objectID)}
type="button"
>
Dismiss
</button>
</span>
</div>
)}
</div>
);
}
}
```

حالا شما سه کامپوننت ES6 دارید. شاید شما متوجه شیء props شده باشید که با استفاده از this از طریق نمونه‌ی کلاس قابل دسترسی است. Props، فرم کوتاه properties (ویژگی)، حاوی همه مقادیری است که در کامپوننت App

خود استفاده کرده‌اید و می‌خواهید در کامپوننت‌های دیگر به آن دسترسی داشته باشید. به این ترتیب، کامپوننت‌ها می‌توانند ویژگی‌ها را به یکدیگر انتقال دهند.

با استخراج این کامپوننت‌ها از کامپوننت App، شما می‌توانید آن‌ها را در جایی دیگر استفاده کنید. از آن‌جا که کامپوننت‌ها با استفاده از شیء props مقادیر خود را می‌گیرند، شما می‌توانید زمانی که از آن‌ها در جایی دیگر استفاده می‌کنید props های مختلف را به کامپوننت خود انتقال دهید. این کامپوننت‌ها قابلیت استفاده مجدد دارند.

تمرین

- کامپوننت‌های بیش‌تری را پیدا کنید که می‌توانید آن‌ها را مانند کامپوننت‌های Search و Table تقسیم کنید.
    - اما اکنون این کار را انجام ندهید، در غیر این صورت در گفتارهای بعد دچار مشکل می‌شوید.

کامپوننت‌های قابل‌ترکیب

یک ویژگی کوچک دیگر وجود دارد که در شیء props قابل دسترسی است: children (فرزند) props. شما می‌توانید از آن برای انتقال دادن المنت‌ها در کامپوننت خودتان از بالا استفاده کنید، که برای خود کامپوننت شناخته شده نیستند، اما امکان ترکیب کامپوننت‌ها به یکدیگر را فراهم می‌کند. بهتر است ببینیم زمانی که فقط یک متن (رشته) را به عنوان یک فرزند به کامپوننت Search انتقال می‌دهید این ویژگی چگونه به نظر می‌رسد.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, list } = this.state;
return (
<div className="App">
<Search
value={searchTerm}
onChange={this.onSearchChange}
>
Search
</Search>
<Table
list={list}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
</div>
);
}
}
```

اکنون کامپوننت Search می‌تواند ویژگی فرزندان را از شیء props جدا کند. سپس می‌تواند مشخص کند که فرزندان کجا باید نمایش داده شوند.

src/App.js

```
class Search extends Component {
render() {
const { value, onChange, children } = this.props;
return (
<form>
{children} <input
type="text"
value={value}
onChange={onChange}
/>
</form>
);
}
}
```

اکنون متن Search باید در کنار فیلد ورودی شما نمایش داده شود. هنگامی که از کامپوننت Search در جایی دیگر استفاده می‌کنید، می‌توانید در صورت تمایل متن متفاوتی را انتخاب کنید. به هر حال، فقط متن نیست که می‌توانید آن را به عنوان فرزند انتقال دهید. شما می‌توانید یک المنت یا نمودار درختی المنت‌ها (که می‌تواند توسط کامپوننت دوباره کپسوله شود) را نیز به عنوان فرزند انتقال دهید. ویژگی فرزندان باعث می‌شود این امکان را داشته باشیم که کامپوننت‌ها را به یکدیگر متصل کنیم.

تمرین

- در مورد مدل‌های ترکیب در ری‌اکت[۴۳] بیش‌تر بخوانید.

[43] https://reactjs.org/docs/composition-vs-inheritance.html

کامپوننت‌های قابل استفاده‌ی مجدد

کامپوننت‌های قابل استفاده‌ی مجدد و ترکیبی شما را قادر می‌سازند به سلسله‌مراتب کامپوننت‌ها را ایجاد کنید. آن‌ها پایه و اساس لایه view ری‌اکت هستند. در گفتار گذشته از اصطلاح قابلیت استفاده‌ی مجدد نام بردیم. شما اکنون می‌توانید کامپوننت‌های table و Search را دوباره استفاده کنید. حتی کامپونت App قابل استفاده‌ی مجدد است، زیرا می‌توانید آن را دوباره در جایی دیگر نشان دهید.

اجازه دهید یک کامپونت قابل استفاده‌ی مجدد، یک کامپونت دگمه، که در نهایت بیش‌تر مورد استفاده‌ی مجدد قرار می‌گیرد را تعریف کنیم.

src/App.js

```
class Button extends Component {
render() {
const {
onClick,
className,
children,
} = this.props;
return (
<button
onClick={onClick}
className={className}
type="button"
>
{children}
</button>
);
}
}
```

ممکن است تعریف چنین کامپونتی بیهوده و اضافی به نظر برسد. شما از یک کامپونت دگمه به جای یک المنت دگمه استفاده می‌کنید. این کار فقط type=”button” را صرفه‌جویی می‌کند. به‌جز ویژگی type، شما باید همه‌ی موارد دیگر را تعریف کنید، هنگامی که می‌خواهید از کامپونت دگمه استفاده کنید؛ اما در این‌جا باید در مورد سرمایه‌گذاری‌های بلندمدت

فکر کنید. تصور کنید که چندین دگمه در اپلیکیشن خود دارید، اما می‌خواهید یک ویژگی، استایل یا رفتار دگمه‌ای را تغییر دهید. بدون استفاده از کامپوننت مجبورید هر دگمه را جداگانه ویرایش کنید. در عوض، کامپوننت دگمه تضمین می‌کند که تنها یک منبع منحصربه‌فرد از حقیقت وجود داشته باشد. یک دگمه برای ویرایش هم‌زمان همه‌ی دگمه‌ها. یک دگمه برای حکومت به همه آن‌ها.

از آن‌جا که از قبل یک المنت دگمه دارید، می‌توانید از کامپوننت دگمه به‌جای آن استفاده کنید. این ویژگی type را حذف می‌کند، زیرا این کامپوننت دگمه است که آن را تعیین می‌نماید.

src/App.js

```
class Table extends Component {
render() {
const { list, pattern, onDismiss } = this.props;
return (
<div>
{list.filter(isSearched(pattern)).map(item =>
<div key={item.objectID}>
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
<span>
<Button onClick={() => onDismiss(item.objectID)}>
Dismiss
</Button>
</span>
</div>
)}
</div>
);
}
}
```

کامپوننت دگمه انتظار وجود یک ویژگی className در props را دارد. ویژگی className یکی دیگر از مشتقات ری‌اکت برای ویژگی Class در HTML است؛ اما وقتی که از دگمه استفاده کرده‌ایم هیچ ClassName به آن انتقال نداده‌ایم. درون کد، باید صریح‌تر ذکر شده باشد که ClassName در کامپوننت دگمه اختیاری است؛ بنابراین شما می‌توانید یک مقدار پیش‌فرض در شیء سازنده‌ی خود اختصاص دهید.

src/App.js

```
class Button extends Component {
render() {
const {
onClick,
className = '',
children,
} = this.props;
...
}
}
```

اکنون، هر بار که در زمان استفاده از کامپوننت دگمه هیچ ویژگی className مشخص نشده باشد، یک رشته‌ی خالی به‌جای ارزش undefined قرار خواهد گرفت.

اعلام کامپوننت‌ها

درحال حاضر شما چهار کامپونت کلاس ES6 دارید؛ اما می‌توانید کار بهتری انجام دهید. اجازه دهید کامپونت‌های فانکشنال بدون state را به عنوان جایگزین برای کامپونت کلاس ES6 معرفی کنم. قبل از این که کامپونت خود را اصلاح کنید، بهتر است انواع کامپونت‌ها را در ری‌اکت معرفی کنیم.

- **کامپونت فانکشنال بدون state**: این کامپونت‌ها توابعی هستند که یک ورودی دریافت می‌کنند و سپس یک خروجی بازمی‌گردانند. ورودی‌ها همان props هستند. خروجی کامپونت یک instance و بنابراین یک JSX ساده است. تا این‌جای کار همه‌چیز کاملاً شبیه به یک کامپونت کلاس ES6 است. با این حال، کامپونت‌های فانکشنال بدون state تابع هستند و در آن‌ها هیچ state لوکالی وجود ندارد. شما نمی‌توانید به State با this.state یا this.setState() دسترسی داشته باشید یا آن را آپدیت کنید، زیرا شیء this وجود ندارد. علاوه‌براین، آن‌ها اصلاً متد چرخه‌ی زندگی ندارند. شما هنوز در مورد متدهای چرخه‌ی زندگی یاد نگرفته‌اید؛ اما قبلاً از دوتا از آن‌ها استفاده کرده‌اید: constructor() و render(). درحالی که سازنده فقط یک بار در طول عمر یک کامپونت اجرا می‌شود، متد کلاس render() یک بار در ابتدا و هربار که کامپونت به‌روزرسانی می‌شود اجرا می‌گردد. زمانی که در یکی از گفتارهای بعد به متدهای چرخه‌ی زندگی می‌رسید، به خاطر داشته باشید که کامپونت فانکشنال بدون state اصلاً متد چرخه‌ی زندگی ندارد.
- **کامپونت‌های کلاس ES6**: شما قبلاً از این نوع کامپونت برای ساخت چهار کامپونت خود استفاده کرده‌اید. در تعریف کلاس، آن‌ها از کامپونت‌های ری‌اکت گسترش می‌یابند. تمام متدهای چرخه‌ی زندگی که در API کامپونت ری‌اکت موجود است، در این کامپونت گسترش می‌یابد. به این ترتیب شما قادر به استفاده از متد کلاس render() شدید. علاوه‌براین شما می‌توانید با استفاده از this.state و this.setState()، stateهای موجود در کامپونت کلاس ES6 را ذخیره و دستکاری کنید.
- **React.create class**: از روش اعلام کامپونت در نسخه‌های قدیمی ری‌اکت استفاده می‌شد و هنوز در ری‌اکت جاوااسکریپت ES5 استفاده می‌شود، اما به دلیل نفع بردن از جاوااسکریپت ES6، فیس‌بوک استفاده از آن را تخلف می‌داند[۴۴]. آن‌ها حتی در نسخه ۱۵٫۵ یک هشدار تخلف اضافه کردند[۴۵]. شما در این کتاب از این روش استفاده نمی‌کنید.

[44] https://reactjs.org/blog/2015/03/10/react-v0.13.html
[45] https://reactjs.org/blog/2017/04/07/react-v15.5.0.html

بنابراین اساساً تنها دو اعلام کامپوننت وجود دارد؛ اما چه موقع از کامپوننت‌های فانکشنال بدون state به‌جای کامپوننت‌های کلاس ES6 استفاده کنیم؟ یک قاعده کلی، استفاده از کامپوننت‌های فانکشنال بدون state در هنگامی‌ست که به state لوکال یا متدهای چرخه‌ی زندگی کامپوننت نیاز ندارید. معمولاً کامپوننت خود به‌عنوان کامپوننت‌های فانکشنال بدون state اجرا می‌کنید. هنگامی که نیاز به دسترسی به stateها و یا متدهای چرخه‌ی زندگی دارید، مجبورید آن‌ها را به شکل کامپوننت کلاس ES6 بازنویسی کنید. ما، به منظور آموزش ری‌اکت، در اپلیکیشن خودمان مسیر دیگری را انتخاب کردیم.

بیایید به اپلیکیشن خود بازگردیم. کامپوننت App از state داخلی استفاده می‌کند. به همین دلیل باید به عنوان کامپوننت کلاس ES6 باقی بماند؛ اما سه مورد دیگر کامپوننت کلاس ES6 شما بدون state هستند. آن‌ها نیازی به this.state یا this.setState() ندارند. حتی بیش‌تر، آن‌ها اصلاً متد چرخه‌ی زندگی ندارند. بیایید کامپوننت Search را به کامپوننت‌های فانکشنال بدون state تبدیل کنیم. بازنویسی کامپوننت Table و دگمه را خودتان به‌عنوان تمرین انجام دهید.

src/App.js

```
function Search(props) {
const { value, onChange, children } = props;
return (
<form>
{children} <input
type="text"
value={value}
onChange={onChange}
/>
</form>
);
}
```

اساس کد به این شکل است. Props ها در امضای تابع قابل دسترسی هستند و مقدار بازگشتی JSX است؛ اما می‌توانید کدهای پیچیده‌تر را در کامپوننت‌های فانکشنال بدون state اجرا کنید. از قبل تغییر ساختار را می‌شناختید.بهترین تمرین استفاده از این روش در امضای تابع برای تغییر ساختار prop ها است.

src/App.js

```
function Search({ value, onChange, children }) {
return (
<form>
{children} <input
type="text"
value={value}
onChange={onChange}
/>
</form>
);
}
```

اما وضعیت می‌تواند از این هم بهتر شود. شما از قبل می‌دانید که تابع arrow در ES6 به شما امکان می‌دهد تا توابع خود را مختصر نگه دارید. می‌توانید بخش بلوک تابع را حذف کنید. در یک بدنه مختصر، یک بازگشت ضمنی ایجاد شده است، بنابراین شما می‌توانید عبارت return را حذف کنید. از آن‌جایی که کامپوننت فانکشنال بدون state شما یک تابع است، شما می‌توانید آن را نیز مختصر نگه دارید.

src/App.js

```
const Search = ({ value, onChange, children }) =>
<form>
{children} <input
type="text"
value={value}
onChange={onChange}
/>
</form>
```

به‌کارگیری آخرین مرحله تنها برای داشتن props به‌عنوان ورودی و JSX به‌عنوان خروجی اهمیت ویژه‌ای داشت. هیچ چیز دیگری در این میان وجود ندارد. با این حال، می‌توانید با استفاده از یک بلوک در تابع arrow ES6 خود، کاری این میان انجام دهید.

Code Playground

```
const Search = ({ value, onChange, children }) => {
// do something
return (
<form>
{children} <input
type="text"
value={value}
onChange={onChange}
/>
</form>
);
}
```

اما اکنون به این کار نیازی ندارید. به همین دلیل است که می‌توانید نسخه‌ی قبلی را بدون بدنه‌ی بلوک نگه دارید. هنگام استفاده از بدنه‌ی بلوک، اغلب افراد تمایل به انجام کارهای بسیار زیادی در تابع دارند. با خروج از بدنه‌ی بلوک، می‌توانید بر روی ورودی و خروجی تابع خود تمرکز کنید.

اکنون یک کامپوننت فانکشنال بدون state کم‌حجم دارید. هنگامی که نیاز به دسترسی به state داخلی کامپوننت یا متدهای چرخه‌ی زندگی داشته باشید، می‌توانید آن را به کامپوننت کلاس ES6 تبدیل کنید. علاوه بر این، شما مشاهده کردید که چگونه می‌توان از جاوااسکریپت ES6 در ری‌اکت استفاده کرد تا آن‌ها را مختصرتر و ظریف‌تر نشان دهد.

تمرین

- کامپوننت‌های Table و Button را به کامپوننت‌های فانکشنال بدون state تبدیل کنید.
- در مورد کامپوننت کلاس ES6 و کامپوننت‌های فانکشنال بدون state[46] بیش‌تر بخوانید.

[46] https://reactjs.org/docs/components-and-props.html

استایل دهی به کامپوننت‌ها

اکنون بهتر است برخی استایل‌های اولیه را به اپلیکیشن و کامپوننت شما اضافه کنیم. می‌توانید فایل‌های src/App.css و src/index.css را دوباره استفاده کنید. این فایل‌ها باید از قبل در پروژه‌ی شما باشند، زیرا پروژه‌ی خود را با create-react-app بوت‌استرپ کرده‌اید. هم‌چنین باید فایل‌های src/App.js و src/index.js نیز در پروژه‌ی شما ایجاد شده باشد. من بعضی از css ها را آماده کرده‌ام که به‌سادگی می‌توانید کپی و در این فایل‌ها ذخیره کنید؛ اما می‌توانید از استایل‌های خودتان هم استفاده کنید.

اول، استایل‌دهی کل اپلیکیشن شما

src/index.css

```
body {
color: #222;
background: #f4f4f4;
font: 400 14px CoreSans, Arial,sans-serif;
} a
{
color: #222;
} a
:hover {
text-decoration: underline;
}
ul, li {
list-style: none;
padding: 0;
margin: 0;
}
input {
padding: 10px;
border-radius: 5px;
outline: none;
margin-right: 10px;
border: 1px solid #dddddd;
}
button {
padding: 10px;
```

```
border-radius: 5px;
border: 1px solid #dddddd;
background: transparent;
color: #808080;
cursor: pointer;
}
button:hover {
color: #222;
} *
:focus {
outline: none;
}
```

دوم، استایل‌دهی به فایل app کامپوننت شما

src/App.css

```
.page {
margin: 20px;
} .
interactions {
text-align: center;
} .
table {
margin: 20px 0;
} .
table-header {
display: flex;
line-height: 24px;
font-size: 16px;
padding: 0 10px;
justify-content: space-between;
} .
table-empty {
margin: 200px;
text-align: center;
```

```
font-size: 16px;
}
```



```
.table-row {
display: flex;
line-height: 24px;
white-space: nowrap;
margin: 10px 0;
padding: 10px;
background: #ffffff;
border: 1px solid #e3e3e3;
} .
table-header > span {
overflow: hidden;
text-overflow: ellipsis;
padding: 0 5px;
} .
table-row > span {
overflow: hidden;
text-overflow: ellipsis;
padding: 0 5px;
} .
button-inline {
border-width: 0;
background: transparent;
color: inherit;
text-align: inherit;
-webkit-font-smoothing: inherit;
padding: 0;
font-size: inherit;
cursor: pointer;
} .
button-active {
border-radius: 0;
border-bottom: 1px solid #38BB6C;
}
```

اکنون می‌توانید استایل‌دهی را در برخی از کامپوننت‌های خود استفاده کنید. فراموش نکنید در ری‌اکت از className به جای ویژگی کلاس در HTML استفاده کنید.

اول، این استایل‌دهی را در کامپوننت کلاس ES6 در App اعمال کنید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, list } = this.state;
return (
<div className="page">
<div className="interactions">
<Search
value={searchTerm}
onChange={this.onSearchChange}
>
Search
</Search>
</div>
<Table
list={list}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
</div>
);
}
}
```

دوم، آن را در کامپوننت‌های فانکشنال بدون state در Table اعمال کنید.

src/App.js

```
const Table = ({ list, pattern, onDismiss }) =>
<div className="table">
{list.filter(isSearched(pattern)).map(item =>
<div key={item.objectID} className="table-row">
<span>
<a href={item.url}>{item.title}</a>
</span>
<span>{item.author}</span>
<span>{item.num_comments}</span>
<span>{item.points}</span>
<span>
Basics in React 79
<Button
onClick={() => onDismiss(item.objectID)}
className="button-inline"
>
Dismiss
</Button>
</span>
</div>
)}
</div>
```

حالا کامپوننت و اپلیکیشن خود را با cssهای اولیه استایل‌دهی کرده‌اید. این کار باید کاملاً مناسب باشد. همان‌طور که می‌دانید، JSX جاوااسکریپت و HTML را با هم ادغام می‌کند. اکنون می‌توان گفت که css را هم به این ترکیب اضافه می‌کند. این عمل، استایل‌دهی inline نامیده می‌شود. شما می‌توانید اشیاء جاوااسکریپت را تعریف کنید و آن را به ویژگی المنت انتقال دهید.

اجازه دهید عرض ستون جدول را با استفاده از استایل‌دهی اینلاین انعطاف‌پذیر کنیم.

src/App.js

```
const Table = ({ list, pattern, onDismiss }) =>
<div className="table">
{list.filter(isSearched(pattern)).map(item =>
```

```
<div key={item.objectID} className="table-row">
<span style={{ width: '40%' }}>
<a href={item.url}>{item.title}</a>
</span>
<span style={{ width: '30%' }}>
{item.author}
</span>
<span style={{ width: '10%' }}>
{item.num_comments}
</span>
<span style={{ width: '10%' }}>
{item.points}
</span>
<span style={{ width: '10%' }}>
<Button
onClick={() => onDismiss(item.objectID)}
className="button-inline"
>
Dismiss
</Button>
Basics in React 80
</span>
</div>
)}
</div>
```

اکنون استایل شما اینلاین است. شما می‌توانید اشیاء استایل را خارج از المنت‌ها تعریف کنید تا تمیزتر شود.

Code Playground

```
const largeColumn = {
width: '40%',
};
const midColumn = {
width: '30%',
};
```

```
const smallColumn = {
width: '10%',
};
```

پس از این کار می‌توانید از آن‌ها در ستون‌های خود استفاده کنید:

به‌طور کل، گزینه‌ها و راه‌حل‌های مختلفی برای استایل‌دهی در ری‌اکت پیدا خواهید کرد. در این‌جا از css محض و استایل‌دهی اینلاین استفاده کرده‌اید. این کار برای شروع کافی است.

من نمی‌خواهم نظر خودم را اعمال کنم، بلکه می‌خواهم شما را با برخی گزینه‌ها تنها بگذارم. می‌توانید در مورد آن‌ها مطالعه کنید و خودتان هم آن‌ها را اعمال کنید؛ اما اگر تازه با ری‌اکت آشنا شده‌اید، توصیه می‌کنم فعلاً از css محض و استایل اینلاین استفاده کنید.

- کامپوننت‌های استایل‌دهی شده[47]
- ماژول‌های css[48]

شما اصول اولیه‌ی نوشتن اپلیکیشن خود را یاد گرفته‌اید! بیایید گفتارهای گذشته را خلاصه‌نویسی کنیم:

- ری‌اکت
  - استفاده از this.state() و setState() برای مدیریت state کامپوننت خود
  - انتقال دادن تابع یا متد کلاس‌ها به المنت هندلر
  - استفاده از فرم‌ها و event ها در ری‌اکت برای اضافه کردن تعاملات
  - جریان داده‌ی یک‌طرفه یک مفهوم مهم در ری‌اکت است
  - کامپوننت‌های کنترل شده را در نظر بگیرید
  - کامپوننت‌های کامپوزیت با فرزندان و کامپوننت‌های قابل استفاده‌ی مجدد
  - استفاده و پیاده‌سازی کامپوننت کلاس ES6 و کامپوننت‌های فانکشنال بدون state
  - رویکردهای مربوط به استایل در کامپوننت شما
- ES6

[47] https://github.com/styled-components/styled-components
[48] https://github.com/css-modules/css-modules

- توابعی که به یک کلاس متصل هستند، متدهای کلاس نامیده می‌شوند
- بازنویسی اشیاء و آرایه‌ها
- پارامترهای پیش‌فرض

- عمومی
  - توابع مرتبه‌ی بالاتر

در این‌جا باز هم یک استراحت منطقی به نظر می‌رسد. آموزش‌ها را کامل یاد بگیرید و آن‌ها را برای خودتان اعمال کنید. می‌توانید با کدهایی که تاکنون نوشته‌اید دست به آزمایش بزنید. شما می‌توانید سورس کد را در منبع رسمی[۴۹] پیدا کنید.

[49] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.2

دریافت واقعی با یک API

اکنون وقت آن است که با یک API واقعی روبرو شویم، زیرا کار کردن با دیتای نمونه می‌تواند خسته کننده باشد.

اگر شما با API ها آشنا نیستید، توصیه می‌کنم حتماً «سفر من برای شناخت API ها» را بخوانید[۵۰].

آیا شما پلتفرم Hacker News[۵۱] را می‌شناسید؟ این پلتفرم یک خبرنامه‌ی عالی در مورد موضوعات مختلف فناوری است. در این کتاب، برای بدست آوردن داستآن‌های محبوب روز از درون این پلتفرم، از API های هکرنیوز استفاده می‌کنید. یک API پایه[۵۲] و جست‌وجو[۵۳] برای دریافت اطلاعات از پلتفرم وجود دارد. مورد دوم برای جست‌وجوی داستآن‌های مربوط به هکرنیوز، در این اپلیکیشن منطقی به نظر می‌رسد. برای درک ساختار داده‌ها می‌توانید مشخصات API را ببینید.

50 https://www.robinwieruch.de/what-is-an-api-javascript/
51 https://news.ycombinator.com
52 https://github.com/HackerNews/API
53 https://hn.algolia.com/api

متدهای چرخه‌ی زندگی

قبل از این که بتوانید داده‌ها را در کامپوننت خود با استفاده از API شروع کنید، باید در مورد متدهای چرخه‌ی زندگی بدانید. این متدها یک hook به چرخه‌ی زندگی کامپوننت‌های ری‌اکت هستند. آن‌ها می‌توانند در کامپوننت‌های کلاس ES6 مورد استفاده قرار بگیرند، اما نه در کامپوننت‌های فانکشنال بدون state.

آیا به یاد می‌آورید که در یکی از گفتارهای قبلی در مورد کلاس‌های جاوااسکریپت ES6 و نحوه‌ی استفاده از آن‌ها در ری‌اکت یاد گرفتید؟ به غیر از متد ()render متدهای متعددی وجود دارد که می‌تواند در یک کامپوننت کلاس ES6 ری‌اکت استفاده شود. همه آن‌ها متدهای چرخه‌ی زندگی هستند. بهتر ااست نگاهی به آن‌ها بیاندازیم:

پیش از این با دو متد چرخه‌ی زندگی آشنا شده‌اید که می‌توان از آن‌ها در کامپوننت کلاس ES6 استفاده کرد: ()render و ()constractor. سازنده فقط زمانی استفاده می‌شود که یک نمونه از کامپوننت ساخته شده و وارد DOM می‌گردد. کامپوننت نمونه‌سازی خواهد شد. این فرآیند نصب کامپوننت نام دارد.

- ()ComponentDidMount: این کامپوننت فقط یک بار در زمان نصب کامپوننت فراخوانی می‌شود. این، مناسب‌ترین زمان برای انجام یک درخواست غیرهم‌زمان برای جمع‌آوری داده‌ها از یک API است. داده‌های جمع‌آوری‌شده در state کامپوننت واقعی ذخیره می‌شوند و در متد چرخه‌ی زندگی ()render نمایش داده می‌شوند.
- (ComponentWillReciveProps(nextProps: این متد چرخه‌ی زندگی در طول به‌روزرسانی چرخه‌ی زندگی فراخوانی می‌شود. به عنوان ورودی props بعدی را دریافت می‌کنید. شما می‌توانید برای اعمال یک رفتار متفاوت بر اساس تفاوت‌های موجود، با استفاده از this.props، props قبلی را با props بعدی مقایسه کنید. علاوه‌براین می‌توانید state را بر اساس props بعدی تنظیم نمایید.
- (ShouldComponentUpdate(nextProps, nextState: این متد چرخه‌ی زندگی بلافاصله قبل از متد ()render فراخوانی می‌شود. شما از قبل props بعدی و state بعدی را در اختیار دارید. می‌توانید از این متد به عنوان آخرین فرصت برای انجام مقدمات قبل از اجرای متد رندر استفاده کنید. توجه داشته باشید که دیگر نمی‌توانید از ()setState استفاده کنید. اگر می‌خواهید state را بر پایه props بعدی محاسبه کنید، باید از ComponentWillReciveProps استفاده کنید.

- ComponentDidUpdate(prevProps,prevState): این متد چرخه‌ی زندگی بلافاصله پس از متد رندر فراخوانی می‌شود. می‌توانید از آن به عنوان فرصتی برای انجام عملیات Dom یا برای انجام درخواست‌های غیرهم‌زمان بعدی استفاده کنید.
- ComponentWillMount: این کامپوننت پیش از آن که کامپوننت شما از بین رود فراخوانی می‌شود. شما می‌توانید از این متد چرخه‌ی زندگی برای انجام هرگونه کارهای پاک کردن (clean up) استفاده کنید.

[پی نوشت مترجم: این کامپوننت در نسخه‌های فعلی ری‌اکت پشتیبانی نمی‌شود و منسوخ شده است.]

متدهای ()Constractor و ()render متدهایی هستند که قبلاً از آن‌ها استفاده کرده‌اید. این متدها معمول‌ترین استفاده‌های متدهای چرخه‌ی زندگی در کامپوننت کلاس ES6 هستند. در واقع استفاده از متد رندر ضروری است، در غیر این صورت شما خروجی کامپوننت نمونه را نخواهید دید.

یک متد چرخه‌ی زندگی دیگر هم وجود دارد: ComponentDidCatch(error, info). این متد در اولین بار در ری‌اکت ۱۶[۵۴] استفاده شد و برای catch خطاها در کامپوننت استفاده می‌شود. به عنوان مثال، نمایش لیست نمونه در اپلیکیشن شما کاملاً خوب کار می‌کند؛ اما ممکن است زمانی که لیست لوکال به صورت اتفاقی null شده باشد، یک مشکل وجود داشته باشد (به عنوان مثال هنگام گرفتن لیست از یک API خارجی، در حالی که درخواست fail شده و شما state لوکال لیست را null تنظیم کرده‌اید). پس از آن، امکان فیلتر کردن و map کردن لیست‌ها وجود نخواهد داشت، زیرا آن لیست null است و نه یک لیست خالی. کامپوننت اینجا درست کار نخواهد کرد و تمام اپلیکیشن fail خواهد شد. در این‌جا با استفاده از ()componentDidCatch می‌توانید خطا را دریافت کنید، آن را در state لوکال خود ذخیره کنید، و یک پیغام اختیاری را برای کاربر اپلیکیشن نمایش دهید که خطایی رخ داده است.

تمرین:

- در مورد متدهای چرخه‌ی زندگی در ری‌اکت[۵۵] بیش‌تر بخوانید.
- در مورد state مربوط به متدهای چرخه‌ی زندگی در ری‌اکت[۵۶] بیش‌تر بخوانید.
- در مورد error handling در ری‌اکت[۵۷] بیش‌تر بخوانید.

[54] https://www.robinwieruch.de/what-is-new-in-react-16/
[55] https://reactjs.org/docs/react-component.html
[56] https://reactjs.org/docs/state-and-lifecycle.html
[57] https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html

دریافت داده‌ها

اکنون شما آماده‌ی دریافت داده‌ها از API هکر نیوز هستید. یک متد چرخه‌ی زندگی وجود دارد که می‌تونید از آن برای دریافت داده‌ها استفاده کنید: componentDidMount(). شما از دریافت native fetch API در جاوااسکریپت برای اجرا کردن درخواست استفاده خواهید کرد. قبل از این که بتوانیم از آن استفاده کنیم، اجازه دهید ثابت‌های URL را برای تقسیم درخواست API به قطعات کوچک‌تر، تنظیم و پارامترهای پیش‌فرض را تعریف کنیم.

src/App.js

```
import React, { Component } from 'react';
import './App.css';
const DEFAULT_QUERY = 'redux';
const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
...
```

در جاوااسکریپت ES6، شما می‌توانید از رشته های قالب (template string)[58] برای ترکیب رشته‌ها استفاده کنید. از این رشته‌ها برای پیوند url خود به نقطه پایانیAPI استفاده خواهید کرد.

Code Playground

```
// ES6
const url = `${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${DEFAULT_QUERY}`;
// ES5
var url = PATH_BASE + PATH_SEARCH + '?' + PARAM_SEARCH + DEFAULT_QUERY;
console.log(url);
// output: https://hn.algolia.com/api/v1/search?query=redux
```

[58] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

این امر باعث می‌شود ترکیب URL شما در آینده انعطاف‌پذیر باشد.

اما اجازه دهید به بحث درخواست API برسیم، جایی که در آن شما می‌خواهید از API استفاده کنید. کل پروسه‌ی جمع‌آوری داده‌ها در یک مرحله ارائه می‌شود. اما هر مرحله در ادامه توضیح داده خواهد شد.

src/App.js

```
...
class App extends Component {
constructor(props) {
super(props);
this.state = {
result: null,
searchTerm: DEFAULT_QUERY,
};
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.onDismiss = this.onDismiss.bind(this);
}
setSearchTopStories(result) {
this.setState({ result });
}
componentDidMount() {
const { searchTerm } = this.state;
fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}`)
.then(response => response.json())
.then(result => this.setSearchTopStories(result))
.catch(error => error);
}
...
}
```

در این کد اتفاقات زیادی می‌افتد. ابتدا در مورد شکستن آن به قطعات کوچک فکر کردم. اما پس از آن برای فهم ارتباط بین هر قطعه دچار مشکل می‌شدید. بنابراین بگذارید هر مرحله را با جزئیات توضیح دهم.

اول، شما می‌توانید لیست نمونه آیتم‌ها را حذف کنید، زیرا یک لیست واقعی از API هکرنیوز بازخواهید گرداند. داده‌های نمونه دیگر مورد استفاده قرار نمی‌گیرند. State اولیه از کامپوننت شما اکنون خالی و Search درحال حاضر پیش‌فرض است. همان عبارت پیش‌فرض Search به عنوان فیلد ورودی کامپوننت Search شما و اولین درخواست شما استفاده می‌شود.

دوم، شما از متد چرخه‌ی زندگی componentDidMount() برای دریافت داده‌ها پس از نصب کامپوننت استفاده می‌کنید. در اولین مرحله، عبارت پیش‌فرض Search از state لوکال استفاده می‌شود. این عبارت داستآن‌های مربوط به "redux" را دریافت خواهد کرد. زیرا آن یک پارامتر پیش‌فرض است.

سوم، API دریافتی اولیه مورد استفاده قرار می‌گیرد. رشته‌های قالب جاوااسکریپت ES6، اجازه‌ی ترکیب url با SearchTerm را می‌دهند. url به عنوان آرگومان برای تابع API دریافتی اولیه عمل می‌کند. پاسخ داده‌شه باید به یک ساختار داده‌ی json تبدیل شود، که در زمان ارتباط با ساختار داده‌ی json، یک مرحله الزامی در تابع API دریافتی اولیه است؛ و در نهایت می‌تواند به عنوان نتیجه در state کامپوننت داخلی تنظیم شود. علاوه‌براین بلوک دریافتی در زمان بروز خطا استفاده می‌شود. اگر در طول درخواست خطایی رخ دهد، این تابع، بلوک دریافتی را به جای بلوک بعدی اجرا می‌کند. در یکی از گفتارهای بعدی کتاب به خطایابی خواهیم پرداخت.

در نهایت، فراموش نکنید متد کامپوننت جدید خود را در سازنده ادغام کنید.

اکنون می‌توانید از داده‌های دریافت‌شده به جای لیست آیتم‌های نمونه استفاده کنید. با این حال، دوباره باید مراقب باشید. نتیجه فقط لیستی از اطلاعات نیست، بلکه یک شیء مرکب با فراداده‌ها و یک لیست از نکاتی است که در کتاب ما همان داستآن‌ها هستند[۵۹]. شما می‌توانید state داخلی را با console.log(this.state) خروجی بگیرید؛ می‌توانید از متد render() برای مشاهده آن استفاده کنید. در مرحله‌ی بعد، شما از خروجی آن برای رندر استفاده می‌کنید. اما ما از رندر شدن هرچیزی جلوگیری خواهیم کرد. بنابراین ما null را برگشت می‌دهیم، در صورتی‌که هیچ نتیجه‌ای در وهله‌ی اول وجود ندارد. هنگامی که درخواست به API موفقیت‌آمیز شد، نتیجه در state لوکال ذخیره خواهد شد و کامپوننت App دوباره با state به‌روزرسانی شده دوباره اجرا می‌شود.

src/App.js

```
class App extends Component {
...
```

[59] https://hn.algolia.com/api

```
render() {
const { searchTerm, result } = this.state;
if (!result) { return null; }
return (
<div className="page">
...
<Table
list={result.hits}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
</div>
);
```

بیایید خلاصه کنیم چه اتفاقی در طول چرخه‌ی زندگی کامپوننت افتاده است. کامپوننت شما توسط سازنده مقداردهی اولیه شده است. پس از آن برای اولین بار رندر می‌شود. اما شما از نمایش هرچیزی جلوگیری می‌کنید، زیرا نتیجه در state لوکال null است. در کامپوننتی که هیچ چیز برای نمایش دادن ندارد، اجازه داده می‌شود تا null برگردانده شود. سپس متد چرخه‌ی زندگی ()componentDidMount اجرا می‌شود. در این متد شما داده‌ها را از API هکرنیوز به صورت ناهمگام دریافت می‌کنید. هنگامی که داده‌ها وارد می‌شوند، state کامپوننت داخلی شما در ()setSearchTopDtories تغییر می‌کند.

پس از آن چرخه‌ی زندگی به‌روزرسانی می‌شود زیرا State لوکال به‌روز شده است. کامپوننت توسط متد ()render دوباره اجرا می‌شود، اما این بار نتیجه‌ی جمع‌آوری شده را در state داخلی شما نگه می‌دارد. کل این کامپوننت، و در نتیجه کامپوننت Table با تمام محتوای آن رندر می‌شوند.

شما از API دریافتی لوکال استفاده کردید که این توسط اکثر مرورگرها برای انجام یک درخواست ناهمگام به یک API پشتیبانی می‌شود. پیکربندی create-react-app اطمینان حاصل می‌کند که در هر مرورگر پشتیبانی می‌شود.

بخش سوم پکیج node هستند که شما می‌توانید از آن برای جایگزینی APIدریافتی اولیه استفاده کنید: axios[60] و superagent[61] .

[60] https://github.com/axios/axios

به یاد داشته باشید که این کتاب از مختصر نویسی جاوااسکریپت برای بررسی درست بودن استفاده می‌کند. در مثال قبلی، if(!result) به‌جای if(result===null) استفاده می‌شود. همین امر برای موارد دیگر در سراسر کتاب نیز اعمال می‌شود. به عنوان مثال، if(!list.length) به‌جای if(list.lenght===0) و یا if(someString) به‌جای (someString!=='') استفاده می‌شود. اگر با این موضوع آشنایی ندارید، در مورد آن مطالعه کنید.

به اپلیکیشن خود بازمی‌گردیم. اکنون لیست hitsها قابل مشاهده است. با این حال، این‌جا دو اشکال در اپلیکیشن وجود دارد. اول، دگمه «رد کردن» خراب است. این دگمه در مورد نتیجه‌ی مرکب شیء چیزی نمی‌داند و هنوز از طریق رد کردن یک آیتم در لیست ساده عمل می‌کند.

دوم، هنگامی که لیست نمایش داده می‌شود، اما شما سعی می‌کنید چیز دیگری را جست‌وجو کنید، لیست برروی client side فیلتر می‌شود. حتی اگر جست‌وجوی اولیه برای جست‌وجوی داستان‌ها روی server-side ساخته شده باشد. رفتار درست می‌تواند دریافت یک شیء نتیجه از API در زمان استفاده از کامپوننت Search باشد.

رفع هر دوی این باگ‌ها در گفتار بعدی انجام خواهد شد.

تمرین

درمورد رشته‌های قالب ES6[۶۲] بیش‌تر بخوانید.

در مورد API دریافتی اولیه[۶۳] بیش‌تر بخوانید.

درمورد دریافت داده‌ها در ری‌اکت[۶۴] بیش‌تر بخوانید.

---

61 https://github.com/visionmedia/superagent
62 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
63 https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
64 https://www.robinwieruch.de/react-fetching-data/

اپراتورهای گسترش ES6

دگمه‌ی رد کردن کار نمی‌کند. چون متد ()onDismiss از شیء نتیجه‌ی ترکیب آگاه نیست.

این دگمه فقط در مورد یک لیست ساده در state محلی می‌داند. اما این دیگر یک لیست ساده نیست. بیایید آن را عوض کنیم تا به‌جای لیست خودش بر روی شیء نتیجه کار کند.

src/App.js

```
onDismiss(id) {
const isNotId = item => item.objectID !== id;
const updatedHits = this.state.result.hits.filter(isNotId);
this.setState({
...
});
}
```

اما اکنون در ()SetState چه اتفاقی می افتد؟ متأسفانه نتیجه یک شیء مرکب «یا پیچیده» است. لیست بازدیدها (hits) فقط یکی از چند ویژگی در شیء است. با این حال، فقط لیست به‌روز می‌شود. زمانی که یک آیتم از شیء نتیجه حذف می‌شود و سایر ویژگی‌ها در همان حالت «بدون تغییر» باقی می مانند.

یک نگرش می‌تواند بازدیدها را در شیء نتیجه تغییر دهد. من آن را نشان خواهم داد. اما ما این کار را به این روش انجام نخواهیم داد.

Code Playground

```
// don`t do this
this.state.result.hits = updatedHits;
```

ری‌اکت شامل ساختار تغییرناپذیر داده‌هاست. بنابراین شما نباید یک شیء را تغییر دهید. (یا به‌طور مستقیم state را تغییر دهید). یک رویکرد بهتر این است که بر اساس اطلاعاتی که دارید، یک شیء جدید ایجاد کنید. در نتیجه هیچ‌یک از اشیاء تغییر نمی‌کند. شما ساختارهای داده تغییر ناپذیر را حفظ خواهید کرد. همیشه یک شیء جدید را بر می‌گردانید و هرگز شیئی را تغییر نخواهید داد.

بنابراین می‌توانید از object assign جاوااسکریپت ES6 استفاده کنید. اولین آرگومان شیء هدف است. تمام آرگومان‌های زیر، اشیاء سورس هستند. این اشیاء به شیء هدف وصل می‌شوند. شیء هدف می‌تواند یک شیء خالی باشد. این شیء تغییرناپذیری را شامل می‌شود، زیرا هیچ شیء سورسی تغییر نمی‌کند. این فرایند شبیه کد زیر خواهد بود:

Code Playground

```
const updatedHits = { hits: updatedHits };
const updatedResult = Object.assign({}, this.state.result, updatedHits);
```

اشیاء بعدی، وقتی با همان نام‌های ویژگی‌ها به‌اشتراک گذاشته می‌شوند، اشیاء ادغام‌شده‌ی قبلی را نادیده می‌گیرند. اکنون این کار را در متد ()onDismiss انجام دهید.

src/App.js

```
onDismiss(id) {
const isNotId = item => item.objectID !== id;
const updatedHits = this.state.result.hits.filter(isNotId);
this.setState({
result: Object.assign({}, this.state.result, { hits: updatedHits })
});
```

{این کد راه حل شما خواهد بود. اما یک روش ساده‌تر در جاوااسکریپت ES6 وجود دارد. آیا مایلید اپراتور گسترش را به شما معرفی کنم؟ این اپراتور فقط از یک سه‌نقطه تشکیل می‌شود... هنگامی که از آن استفاده می‌شود، تمام مقادیر از یک آرایه یا یک شیء به یک آرایه یا یک شیء دیگر کپی می‌شود.

بیایید اپراتور گسترش آرایه ES6 را بررسی کنیم حتی اگر هنوز به آن نیازی ندارید.

Code Playground

```
const userList = ['Robin', 'Andrew', 'Dan'];
const additionalUser = 'Jordan';
const allUsers = [ ...userList, additionalUser ];
console.log(allUsers);
// output: ['Robin', 'Andrew', 'Dan', 'Jordan']
```

متغیر allUser یک آرایه کاملاً جدید است. متغیرهای userlist و additionalUser به همان شکل باقی می‌مانند. حتی می‌توانید دو آرایه به یک آرایه تبدیل کنید.

Code Playground

```
const oldUsers = ['Robin', 'Andrew'];
const newUsers = ['Dan', 'Jordan'];
const allUsers = [ ...oldUsers, ...newUsers ];
console.log(allUsers);
// output: ['Robin', 'Andrew', 'Dan', 'Jordan']
```

حالا بیایید نگاهی به اپراتور گسترش شیء بیندازیم. این جاوااسکریپت ES6 نیست. این یک پیشنهاد برای نسخه‌ی بعدی جاوااسکریپت[65] است که در حال حاضر توسط جامعه‌ی ری‌اکت استفاده شده است. به همین دلیل است که create- react- app این ویژگی را در پیکربندی ایجاد کرده است.

[65] https://github.com/tc39/proposal-object-rest-spread

اساساً این نسخه همانند اپراتور گسترش‌دهنده‌ی آرایه‌ی جاوااسکریپت ES6 است، اما حاوی اشیاء نیز هست. این نسخه تمام مقادیر key را در یک شیء جدید کپی می‌کند.

Code Playground

```
const userNames = { firstname: 'Robin', lastname: 'Wieruch' };
const age = 28;
const user = { ...userNames, age };
console.log(user);
// output: { firstname: 'Robin', lastname: 'Wieruch', age: 28 }
```

اشیاء چندگانه می‌توانند همانند مثال گسترش‌یافته‌ی آرایه ایجاد شوند.

Code Playground

```
const userNames = { firstname: 'Robin', lastname: 'Wieruch' };
const userAge = { age: 28 };
const user = { ...userNames, ...userAge };
console.log(user);
// output: { firstname: 'Robin', lastname: 'Wieruch', age: 28 }
```

در نهایت، می توان از آن برای جایگزینی ()object.assign استفاده کرد.

src/App.js

```
onDismiss(id) {
const isNotId = item => item.objectID !== id;
const updatedHits = this.state.result.hits.filter(isNotId);
this.setState({
result: { ...this.state.result, hits: updatedHits }
});
}
```

اکنون دگمه‌ی «رد کردن» باید دوباره کار کند. زیرا متد ()onDismiss از شیء نتیجه‌ی مرکب و چگونگی به‌روز کردن آن بعد از رد کردن یک آیتم از لیست، باخبر است.

تمرین

- در مورد object.assign() در ES6[۶۶] بیش‌تر بخوانید.
- در مورد اپراتور گسترش آرایه در ES6[۶۷] بیش‌تر بخوانید.

[66] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign

- اپراتور گسترش شیء به‌طور خلاصه ذکر شده است.

[67] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

رندر شرطی

رندر شرطی در اولین اپ‌های ری‌اکت معرفی شده است. اما این موضوع در مورد این کتاب صادق نیست، زیرا هنوز مورد استفاده آن وجود نداشته است. رندر شرطی زمانی اتفاق می افتد که می خواهید یک یا چند المنت را رندر کنید. گاهی اوقات این بدان معناست که یک المنت یا رندر کند یا نکند. به هر حال، ساده ترین استفاده رندر شرطی می‌تواند با یک عبارت if-else در Jsx بیان شود.

شیء نتیجه در state کامپوننت داخلی در ابتدا null است. تاکنون کامپوننت APP هیچ المنتی را دریافت نکرده است. وقتی نتیجه، هیچ چیز از API دریافت نکرده است. این خودش یک رندر شرطی است. زیرا قبلاً از متد چرخه‌ی زندگی ()render، از رندر برای یک شرط خاص استفاده می‌کنید. کامپوننت APP یا هیچ چیزی را رندر نمی‌کند، و یا آن المنت را ارائه می‌دهد.

اما بیایید یک قدم جلوتر برویم. بیایید کامپوننت جدول، که تنها کامپوننتی است که به نتیجه بستگی دارد، را در یک رندر شرطی مستقل ایجاد کنیم. می‌توانید به‌راحتی از اپراتور سه‌جانبه در JSX خود استفاده کنید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, result } = this.state;
return (
<div className="page">
<div className="interactions">
<Search
value={searchTerm}
onChange={this.onSearchChange}
>
Search
</Search>
</div>
{ result
? <Table
list={result.hits}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
: null
}
Getting Real with an API 94
</div>
);
}
}
```

این دومین گزینه برای بیان یک رندر شرطی است. گزینه‌ی سوم اپراتور منطقی && است. در جاوااسکریپت یک && “Hello World” درست همیشه با “Hello World” ارزیابی می‌شود. و یک && “Hello World” غلط همیشه با && ارزیابی می‌شود.

Code Playground

```
const result = true && 'Hello World';
console.log(result);
// output: Hello World
const result = false && 'Hello World';
console.log(result);
// output: false
```

در ری‌اکت می‌توانید از این رفتار استفاده کنید. اگر شرط درست باشد، عبارت بعد از اپراتور منطقی && خروجی خواهد بود و اگر نادرست باشد، ری‌اکت آن را نادیده گرفته و عبارت از بین می‌رود. این فرایند برای رندر شرطی Table مناسب است. زیرا باید یا Table را بازگرداند و یا هیچ چیز را تغییر ندهد.

src/App.js

```
{ result &&
<Table
list={result.hits}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
}
```

این‌ها چند روش برای استفاده از رندر شرطی در ری‌اکت بودند. شما می‌توانید در مورد گزینه‌های بیش‌تر در لیست کامل لیست‌یابی از رویکردهای رندر شرطی[۶۸] بخوانید.

علاوه بر این، به هر حال موارد استفاده‌ی مختلف و زمان استفاده از آن‌ها را خواهید دانست. شما باید قادر به دیدن داده‌های گرفته‌شده در اپ خود باشید. همه چیز به جز Table باید نمایش داده شود. در طی داده‌های گرفته‌شده هنگامی که درخواست نتیجه را اعمال می‌کند و داستان در state محلی قرار دارد، جدول نمایش داده می‌شود، زیرا متد render() دوباره اجرا می‌شود و شرط به نفع نمایش کامپوننت Table در رندر شرطی رفع می‌گردد.

تمرین:

- در مورد روش‌های مختلف رندر شرطی[۶۹] بیش‌تر بخوانید.
- در مورد رندرهای شرطی در ری‌اکت[۷۰] بیش‌تر بخوانید.

[68] https://www.robinwieruch.de/conditional-rendering-react/
[69] https://www.robinwieruch.de/conditional-rendering-react/

[70] https://reactjs.org/docs/conditional-rendering.html

جست‌وجوی سمت سرور یا مشتری

هنگام استفاده از کامپوننت Search با فیلد ورودی، لیست را فیلتر می‌کنید. اما این اتفاق در سمت مشتری رخ می‌دهد. اکنون قرار است از API هکر نیوز برای جست‌وجو در سمت server استفاده کنید. در غیر این صورت، با پاسخ اولین API که از ()ComponentDidMount با پارامتر عبارت جست‌وجوی پیش‌فرض مواجه خواهید شد.

می‌توانید متد ()onSearchSubmit را در کامپوننت APP خود ایجاد کنید که وقتی جست‌وجو در کامپوننت Search اجرا می‌شود، نتیجه را از API هکر نیوز دریافت کنید.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
result: null,
searchTerm: DEFAULT_QUERY,
};
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.onSearchSubmit = this.onSearchSubmit.bind(this);
this.onDismiss = this.onDismiss.bind(this);
}
...
onSearchSubmit() {
const { searchTerm } = this.state;
}
...
}
```

متد ()onSearchSubmit باید برای همان کارایی که متد چرخه‌ی زندگی ()ComponentDidMount انجام می‌دهد استفاده شود، اما این بار با یک تغییر ترم جست‌وجو از state محلی و نه با ترم جست‌وجوی پیش‌فرض اولیه. بنابراین شما می‌توانید عملکرد را به عنوان یک متد کلاس قابل‌استفاده‌ی مجدد استفاده کنید.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
result: null,
searchTerm: DEFAULT_QUERY,
};
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.fetchSearchTopStories = this.fetchSearchTopStories.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.onSearchSubmit = this.onSearchSubmit.bind(this);
this.onDismiss = this.onDismiss.bind(this);
```

```
}
...
fetchSearchTopStories(searchTerm) {
fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}`)
.then(response => response.json())
.then(result => this.setSearchTopStories(result))
.catch(error => error);
}
componentDidMount() {
const { searchTerm } = this.state;
this.fetchSearchTopStories(searchTerm);
}
...
onSearchSubmit() {
const { searchTerm } = this.state;
this.fetchSearchTopStories(searchTerm);
}
...
}
```

اکنون کامپوننت Search یک دگمه دیگر را اضافه می‌کند. این دگمه درخواست جست‌وجو را منعکس می‌کند. در غیر این صورت شما می‌توانید داده‌ها را هر زمان که فیلد ورودی تغییر می‌کند، از API هکرنیوز دریافت کنید. اما صراحتاً تمایل دارید این کار را با هندلر onClick() انجام دهید.

به عنوان جایگزین می‌توانید debounce (تأخیر) تابع onChange() و دگمه را spare کنید. اما این کار پیچیدگی را افزایش می‌دهد و شاید اثر مطلوب نداشته باشد. بیایید آن را ساده نگه داریم و debounce کردن را فعلاً کنار بگذاریم.

اول ، متد onSearchSubmit() را به کامپوننت search خود انتقال دهید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, result } = this.state;
return (
<div className="page">
<div className="interactions">
<Search
value={searchTerm}
onChange={this.onSearchChange}
onSubmit={this.onSearchSubmit}
>
Search
</Search>
</div>
{ result &&
<Table
```

```
list={result.hits}
pattern={searchTerm}
onDismiss={this.onDismiss}
/>
}
</div>
);
}
}
```

دوم، دگمه‌ای را در کامپوننت Search خود ایجاد کنید. این دگمه "Type="Submit دارد و فرم آن از ویژگی onSubmit() برای انتقال دادن متد onSubmit() استفاده می‌کند. شما می‌توانید دوباره از ویژگی فرزندان استفاده کنید، اما این بار به عنوان محتوای دگمه استفاده می‌شود.

src/App.js

```
const Search = ({
value,
onChange,
onSubmit,
children
}) =>
<form onSubmit={onSubmit}>
<input
type="text"
value={value}
onChange={onChange}
/>
<button type="submit">
{children}
</button>
</form>
```

در Table می‌توانید قابلیت فیلتر را حذف کنید، زیرا فیلتر بیش‌تری سمت سرور (جست‌وجو) وجود نخواهد داشت. فراموش نکنید که تابع isSearched() هم حذف شود. این تابع دیگر استفاده نخواهد شد. این نتیجه پس از کلیک بر روی دگمه «جست‌وجو» به‌طور مستقیم از API هکر نیوز می‌آید.

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, result } = this.state;
return (
<div className="page">
...
```

```
{ result &&
<Table
list={result.hits}
onDismiss={this.onDismiss}
/>
}
</div>
);
}
Getting Real with an API 100
}
...
const Table = ({ list, onDismiss }) =>
<div className="table">
{list.map(item =>
...
)}
</div>
```

اکنون هنگامی که سعی در جست‌وجو می‌کنید، متوجه بارگیری مجدد مرورگر می‌شوید. این یک رفتار اولیه‌ی مرورگر برای ارسال پاسخ در یک فرم HTML است. در ری‌اکت، اغلب از متد preventDefault() برای سرکوب این رفتار مرورگر استفاده می‌کنید.

src/App.js

```
onSearchSubmit(event) {
const { searchTerm } = this.state;
this.fetchSearchTopStories(searchTerm);
event.preventDefault();
}
```

اکنون شما باید قادر به جست‌وجو در داستان‌های هکر نیوز باشید. عالی است، شما با یک API دنیای واقعی تعامل می‌کنید. دیگر نیازی به جست‌وجو در سمت مشتری نیست.

تمرین

- در مورد ایونت‌های ترکیبی در ری‌اکت[71] بیش‌تر بخوانید.
- با API هکر نیوز[72] تجربه کنید.

[71] https://reactjs.org/docs/events.html
[72] https://hn.algolia.com/api

بازپس‌گیری صفحه

آیا تاکنون نگاه دقیق‌تری به ساختار داده‌ی برگشتی داشته‌اید؟ API هکر نیوز[۷۳] چیزهایی بیش از لیست بازدیدها را بازمی‌گرداند. این API دقیقاً یک لیست صفحه‌بندی را بازمی‌گرداند. ویژگی صفحه page، که اولین پاسخ تهی را برمی‌گرداند، می‌تواند برای گرفتن زیر مجموعه صفحه‌بندی، به عنوان نتیجه، مورد استفاده قرار گیرد. شما فقط باید صفحه‌ی بعدی را با همان عبارت جست‌وجو به API انتقال دهید.

بیایید ثابت‌های API مرکب را گسترش دهیم به‌طوری که بتواند با داده‌ی صفحه‌بندی شده ایجاد شود.

src/App.js

```
const DEFAULT_QUERY = 'redux';
const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';
```

اکنون می‌توانید از ثابت جدیدی استفاده کنید تا پارامتر page را به درخواست API خود اضافه کنید.

Code Playground

```
const url = `${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAG\
E}`;
console.log(url);
// output: https://hn.algolia.com/api/v1/search?query=redux&page=
```

متد fetchSearchTopStories() صفحه را به عنوان آرگومان دوم دریافت می‌کند. اگر آرگومان دوم را به آن ندهید، در درخواست اولیه صفر (0) جایگزین خواهد شد. بنابراین متدهای componentDidmount() و onSearchSubmit() با درخواست اول صفحه اول را می‌گیرند. در هر بار اضافه کردن، باید صفحه‌ی بعدی را به عنوان آرگومان دوم وارد کنید.

src/App.js

```
class App extends Component {
...
fetchSearchTopStories(searchTerm, page = 0) {
fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}\
${page}`)
.then(response => response.json())
.then(result => this.setSearchTopStories(result))
.catch(error => error);
}
...
}
```

[73] https://hn.algolia.com/api

آرگومان page از پارامتر پیش‌فرض جاوااسکریپت Es6 استفاده می‌کند تا به page صفر برگردید در صورتی که هیچ آرگومان page برای تابع مشخص نشده است.

اکنون می‌توانید از page جاری از پاسخ API در ()fetchSearchTopStories استفاده کنید. می‌توانید از این متد در یک دگمه استفاده کنید. برای گرفتن داستآن‌های بیش‌تر در هندلر دگمه API اجازه دهید از دگمه استفاده کنیم تا داده‌های صفحه‌بندی بیش‌تری از API از هکر نیوز بیرون بیاوریم. شما تنها نیاز به تعریف هندلر ()onClick دارید که عبارت جست‌وجوی جاری و صفحه بعدی را می گیرد (صفحه جاری+۱).

src/App.js

```
class App extends Component {
...
render() {
const { searchTerm, result } = this.state;
const page = (result && result.page) || 0;
return (
<div className="page">
<div className="interactions">
...
{ result &&
<Table
list={result.hits}
onDismiss={this.onDismiss}
/>
}
<div className="interactions">
<Button onClick={() => this.fetchSearchTopStories(searchTerm, page + 1\
)}>
More
</Button>
</div>
</div>
);
}
}
```

علاوه بر این، در متد ()render شما باید اطمینان حاصل کنید زمانی که هنوز هیچ نتیجه‌ای وجود ندارد، صفحه‌ی پیش‌فرض صفر تعریف شود. به خاطر داشته باشید که متد ()render قبل از این که داده‌ها به صورت ناهمگام در متد چرخه‌ی زندگی ()ComponentDidmount گرفته شوند، فراخوانی شود.

یک مرحله کم شده است. شما صفحه‌ی بعدی داده را می‌گیرید، اما صفحه قبلی داده‌ها را نیز حذف می‌کند. بهترین راه حل، پیوستن لیست جدید و قدیم بازدیدها از state محلی و ایجاد یک شیء نتیجه‌ی جدید است. بیایید برای اضافه کردن داده‌های جدید به جای حذف آن‌ها، عملکرد را تنظیم کنیم.

src/App.js

```
setSearchTopStories(result) {
const { hits, page } = result;
const oldHits = page !== 0
? this.state.result.hits
: [];
const updatedHits = [
...oldHits,
...hits
];
this.setState({
result: { hits: updatedHits, page }
});
}
```

اکنون چند اتفاق در متد setSearchTopStories() اتفاق می‌افتد. اول، شما بازدیدها و صفحه را در نتیجه دریافت می‌کنید.

دوم، شما باید چک کنید آیا بازدیدهای قدیمی قبل از وجود داشته‌اند یا نه. وقتی که صفحه 0 است، یک درخواست جست‌وجوی جدید از componentDidMount() یا onSearchSubmit() وجود دارد. بازدیدها خالی هستند. اما وقتی روی دگمه‌ی «بیش‌تر» کلیک می‌کنید تا داده‌های صفحه‌بندی را دریافت کنید، صفحه دیگر صفر نخواهد بود. این صفحه‌ی بعدی است. اکنون بازدیدهای قدیمی در state شما ذخیره می‌شوند و در نتیجه می‌توانند مورد استفاده قرار گیرند.

سوم، شما نمی‌خواهید بازدید قدیمی را override کنید. شما می‌توانید بازدیدهای قدیمی و جدید را از درخواست API اخیر ترکیب کنید. ادغام دو لیست می‌تواند با اپراتور گسترش آرایه جاوااسکریپت ES6 انجام شود.

چهارم، ترکیب بازدید و صفحه را در state کامپوننت محلی تنظیم کنید.

شما می‌توانید آخرین تنظیمات را انجام دهید. هنگامی که دگمه‌ی «بیش‌تر» را امتحان می‌کنید، فقط چند آیتم از لیست می گیرید. API URL می‌تواند برای دریافت آیتم‌های بیش‌تری از لیست گسترش یابد. باز هم می‌توانید ثابت‌های path ترکیبی بیش‌تری اضافه کنید.

src/App.js

```
const DEFAULT_QUERY = 'redux';
const DEFAULT_HPP = '100';

const PATH_BASE = 'https://hn.algolia.com/api/v1';
const PATH_SEARCH = '/search';
const PARAM_SEARCH = 'query=';
const PARAM_PAGE = 'page=';
const PARAM_HPP = 'hitsPerPage=';
```

اکنون می‌توانید از ثابت برای گسترش API URL استفاده کنید.

src/App.js

```
fetchSearchTopStories(searchTerm, page = 0) {
fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}${\
page}&${PARAM_HPP}${DEFAULT_HPP}`)
.then(response => response.json())
.then(result => this.setSearchTopStories(result))
.catch(error => error);
}
```

سپس درخواست از API هکر نیوز تعداد آیتم‌های بیش‌تری نسبت به گذشته را از لیست در یک روز درخواست برمی‌گرداند. همان‌طور که می‌بینید، یک API قدرتمند مانند API هکر نیوز راه‌های زیادی برای تجربه داده‌های دنیای واقعی به شما می‌دهد. شما باید از آن استفاده کنید تا هنگام یادگیری، بتوانید چیزهای هیجان‌انگیزتر و جدیدتر را امتحان کنید. این فرایند نشان می‌دهد که هنگام یادگیری یک زبان برنامه نویسی جدید یا یک کتابخانه جدید، چگونه در مورد توانایی‌هایی که API ارائه می دهند یاد گرفتم[۷۴].

تمرین

- در مورد پارامترهای پیش‌فرض در ES6[۷۵] بیش‌تر بخوانید.
- با پارامترهای API هکرنیوز[۷۶] آزمایش کنید.

[74] https://www.robinwieruch.de/what-is-an-api-javascript/
[75] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters
[76] https://hn.algolia.com/api

کش مشتری

هر جست‌وجو یک درخواست در API ایجاد می‌کند. شما ممکن است برای redux جست‌وجو کنید پس از آن react و پس از آن دوباره redux را جست‌وجو کنید. این فرایند در کل سه درخواست را می‌سازد. اما برای redux دوبار جست‌وجو کردید، و هر دو بار کل دوره‌ی ناهمگام را برای جمع‌آوری داده‌ها را طی می‌کنید. در یک حافظه سمت مشتری، تمام نتایج را ذخیره می‌کنید. هنگامی که یک درخواست برای API ساخته شده است، آن را بررسی می‌کند که آیا نتیجه از قبل وجود دارد یا نه؛ اگر وجود دارد، از cache استفاده می‌شود. در غیر این صورت، درخواست API برای جمع‌آوری داده‌ها انجام می‌شود.

برای داشتن یک client cache برای هر نتیجه، باید به‌جای یک نتیجه چندین نتیجه را در state کامپوننت داخلی خود ذخیره کنید. شیء نتایج MAP را انجام خواهد داد و عبارت جست‌وجو را به عنوان key و نتیجه را به عنوان مقدار در نظر خواهد گرفت. هر نتیجه از API با عبارت جست‌وجو key ذخیره خواهد شد.

حالا نتیجه‌ی شما در state محلی به شرح زیر می باشد:

Code Playground

```
result: {
hits: [ ... ],
page: 2,
}
```

تصور کنید که شما دو درخواست API ایجاد کرده‌اید، یکی برای جست‌وجوی "Redux" و یکی برای "Redux". شیء نتایج باید شبیه به زیر باشد:

Code Playground

```
results: {
redux: {
hits: [ ... ],
page: 2,
},
react: {
hits: [ ... ],
page: 1,
},
...
}
```

بیایید با استفاده از set state در ری‌اکت، یک cache سمت client را ایجاد کنیم. اول شیء نتیجه را در state کامپوننت داخلی به نتایج تغییر نام دهید؛ دوم با استفاده از ذخیره هر نتیجه، یک searchKey موقتی تعریف کنید.

src/App.js

```
class App extends Component {
constructor(props) {
```

```
super(props);
this.state = {
results: null,
searchKey: '',
searchTerm: DEFAULT_QUERY,
};
...
}
...
}
```

قبل از هر درخواست، searchKey باید تنظیم شود. این امر نشان‌دهنده‌ی searchTerm است. ممکن است تعجب کنید چرا ما در وهله‌ی اول از searchTerm استفاده نمی کنیم. این بخش مهمی است که قبل از پیاده‌سازی باید درک شود. searchTerm یک متغیر در حال نوسان است، زیرا هر بار که در فیلد ورودی جست‌وجو وارد می‌شود، تغییر می‌یابد. با این حال، در نهایت به یک متغیر غیرنوسان‌کننده نیاز خواهید داشت. این بخش جست‌وجوی سابمیت‌شده‌ی ری‌اکت را در API مشخص می‌کند و می‌تواند برای بازیابی نتیجه‌ی درست از MAP نتایج استفاده شود. این فرایند یک اشاره‌گر به نتیجه‌ی اخیر شما در cache است و بنابراین می‌تواند برای نمایش نتیجه‌ی جاری در متد ()render مورد استفاده قرار گیرد.

src/App.js

```
componentDidMount() {
const { searchTerm } = this.state;
this.setState({ searchKey: searchTerm });
this.fetchSearchTopStories(searchTerm);
}
onSearchSubmit(event) {
const { searchTerm } = this.state;
this.setState({ searchKey: searchTerm });
this.fetchSearchTopStories(searchTerm);
event.preventDefault();
}
```

اکنون شما باید عملکردی را تنظیم کنید که نتیجه در state کامپوننت داخلی ذخیره شود. این عملکرد باید هر نتیجه را با searchKey ذخیره کند.

src/App.js

```
class App extends Component {
...
setSearchTopStories(result) {
const { hits, page } = result;
const { searchKey, results } = this.state;
const oldHits = results && results[searchKey]
? results[searchKey].hits
```

```
: [];
const updatedHits = [
...oldHits,
...hits
];
this.setState({
results: {
...results,
[searchKey]: { hits: updatedHits, page }
}
});
}
...
}
```

searchKey به عنوان key برای ذخیره‌ی آپدیت بازدیدها و page در map نتایج استفاده خواهد شد.

اول، شما باید searchKey را از state کامپوننت بازیابی کنید. به خاطر داشته باشید که searchKey باید بر روی ComponentDidmount() یا onSearchSubmit() تنظیم شود.

دوم، بازدیدهای قدیمی باید همانند قبل با بازدید جدید، ترکیب شوند. اما این بار نمایش‌های قدیمی از map نتایج با searchKey به عنوان key قابل بازیابی است.

سوم، یک نتیجه‌ی جدید می‌تواند رد map نتیجه ها در state تنظیم شود.

بیایید شیء نتایج را در set state بررسی کنیم.

src/App.js

```
results: {
...results,
[searchKey]: { hits: updatedHits, page }
}
```

قسمت پایین اطمینان حاصل می‌کند که نتیجه‌ی به‌روز شده با searchKey در map نتایج ذخیره می‌شود. مقدار یک شیء برابر با ویژگی صفحه و تعداد بازدیدها است. searchKey عبارت جست‌وجو است. شما قبلاً در مورد {searchKey } Syntax ... آموخته‌اید. این یک ویژگی محاسبه‌شده‌ی name است. این ویژگی به شما کمک می‌کند تا مقادیر را به صورت پویا در شیء اختصاص دهید.

بخش بالا نیز نیاز به گسترش تمام نتایج دیگر توسط searchKey در state دارد که با استفاده از اپراتور گسترش انجام می‌شود. در غیر این صورت تمام نتایجی که قبلاً ذخیره کرده‌اید را از دست می‌دهید.

اکنون تمام نتایج را با ترم جست‌وجو ذخیره می‌کنید. این اولین گام برای فعال کردن cache شماست. در مرحله‌ی بعد، می‌توانید نتایج را با استفاده از searchKey غیرنوسانی، از map نتایج خود ارزیابی کنید. به همین دلیل است که باید

searchKey را در وهله‌ی اول به عنوان متغیر غیرنوسانی معرفی کنید. در غیر این صورت زمانی که شما از searchTerm دارای نوسان استفاده می‌کنید تا نتیجه کنونی را بازیابی کنید، بازیابی شکسته خواهد شد، چون این مقدار ممکن است هنگام استفاده از کامپوننت search تغییر کند.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey
} = this.state;
const page = (
results &&
results[searchKey] &&
results[searchKey].page
) || 0;
const list = (
results &&
results[searchKey] &&
results[searchKey].hits) || [];
return (
<div className="page">
<div className="interactions">
...
</div>
<Table
list={list}
onDismiss={this.onDismiss}
/>
<div className="interactions">
<Button onClick={() => this.fetchSearchTopStories(searchKey, page + 1)\
}>
More
</Button>
</div>
</div>
);
}
}
```

از آن‌جایی که شما به طور پیش‌فرض یک لیست خالی دارید، زمانی که هیچ نتیجه‌ای توسط searchKey وجود ندارد، می‌توانید اکنون رندر شرطی را برای کامپوننت Table چشم‌پوشی کنید. علاوه بر این، باید searchKey را به جای SearchTerm دگمه‌ی «بیش‌تر» انتقال دهید.

در غیر این صورت گرفتن صفحه‌بندی شما به مقدار SearchTerm دارای نوسان بستگی دارد. علاوه بر این اطمینان حاصل کنید که ویژگی SearchTerm دارای نوسان را برای فیلد ورودی در کامپوننت Search نگه می‌دارید.

قابلیت جست‌وجو باید دوباره کار کند، این همه نتایج از API هکرنیوز را ذخیره می‌کند.

علاوه بر این، متد ()onDismiss نیاز به بهبود دارد. این متد هنوز با شیء نتیجه سروکار دارد. اکنون باید با چند نتیجه روبرو شویم.

src/App.js

```
onDismiss(id) {
const { searchKey, results } = this.state;
const { hits, page } = results[searchKey];
const isNotId = item => item.objectID !== id;
const updatedHits = hits.filter(isNotId);
this.setState({
results: {...results,
[searchKey]: { hits: updatedHits, page }
}
});
}
```

دگمه‌ی «رد کردن» باید دوباره کار کند.

با این حال، هیچ چیز اپ را از ارسال یک درخواست API در هر سابمیت جست‌وجو متوقف نمی‌کند. اگرچه ممکن است نتیجه قبلاً وجود داشته باشد، اما هیچ بررسی‌ای وجود ندارد که از درخواست جلوگیری کند. بنابراین قابلیت cache هنوز کامل نشده است. این فرایند نتایج را ذخیره می‌کند، اما از آن‌ها استفاده نمی‌کند. آخرین مرحله ممکن است هنگامی که یک نتیجه در cache در دسترس باشد از درخواست API جلوگیری کند.

src/App.js

```
class App extends Component {
constructor(props) {
...
this.needsToSearchTopStories = this.needsToSearchTopStories.bind(this);
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.fetchSearchTopStories = this.fetchSearchTopStories.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.onSearchSubmit = this.onSearchSubmit.bind(this);
this.onDismiss = this.onDismiss.bind(this);
}
needsToSearchTopStories(searchTerm) {
return !this.state.results[searchTerm];
}
...
onSearchSubmit(event) {
const { searchTerm } = this.state;
this.setState({ searchKey: searchTerm });
```

```
if (this.needsToSearchTopStories(searchTerm)) {
this.fetchSearchTopStories(searchTerm);
}
event.preventDefault();
}
...
}
```

اکنون مشتری شما فقط یک بار درخواست API را ارسال می‌کند. هرچند برای یک کلمه دوبار جست‌وجو کنید. حتی داده‌های صفحه‌بندی شده با چندین صفحه به این شیوه cache می‌شوند، زیرا شما همیشه آخرین page را برای هر نتیجه در map نتایج ذخیره می‌کنید. آیا این رویکردی قدرتمند برای معرفی cache در اپ شما نیست؟ API هکر نیوز شما را به همه‌ی چیزهایی که نیاز دارید، برای cache مجهز می‌کند.

مدیریت ارورها

همه چیز برای تعاملات شما با API هکر نیوز حاضر است. شما حتی یک راه دقیق برای cache نتایج از API و استفاده از قابلیت صفحه‌بندی برای آوردن یک لیست بی‌پایان از سابمیت داستآن‌ها از API را تعریف کرده‌اید. اما یک چیز فراموش شده است. متأسفانه اغلب برنامه‌های درحال‌توسعه این مسأله را فراموش می‌کنند: مدیریت ارورها. اجرای یک الگوی درست بدون نگرانی در مورد خطاهایی که ممکن است در طول مسیر اتفاق بیافتد، بسیار آسان است. در این گفتار شما یک راه حل مؤثر برای اضافه کردن مدیریت ارورها برای اپ خود در صورتی که درخواست API با خطا مواجه شود را یاد می‌گیرید. قبلاً در مورد ضرورت ساخت بلوک ها در ری‌اکت برای معرفی خطاها یاد گرفته‌اید: state محلی و رندرینگ شرطی. اساساً خطا فقط یک حالت دیگر در ری‌اکت است. هنگامی که یک خطا رخ می‌دهد، شما آن را در state محلی ذخیره کرده و در کامپوننت خود با رندر شرطی نمایش می‌دهید. بیایید یک خطا را در کامپوننت APP اجرا کنیم، زیرا این کامپونتی است که در مرحله‌ی اول برای گرفتن داده‌ها از API هکر نیوز استفاده شده است. ابتدا، شما باید خطا را در state محلی وارد کنید. این فرایند به صورت null مقداردهی می‌شود، اما در صورت ایجاد ارور، به شکل شیء ارور تنظیم می‌شود.

src/App.js

```
class App extends Component {
constructor(props) {
super(props);
this.state = {
results: null,
searchKey: '',
searchTerm: DEFAULT_QUERY,
error: null,
};
...
}
...
}
```

دوم، می‌توانید از بلوک cache در native fetch خود برای ذخیره شیء ارور در state محلی استفاده کنید. با استفاده از set state ، هربار که درخواست API ناموفق بود، بلوک cache اجرا می‌شود.

src/App.js

```
class App extends Component {
...
fetchSearchTopStories(searchTerm, page = 0) {
fetch(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}\
${page}&${PARAM_HPP}${DEFAULT_HPP}`)
.then(response => response.json())
.then(result => this.setSearchTopStories(result))
.catch(error => this.setState({ error }));
}
...
}
```

سوم، می‌توانید شیء ارور را در متد ()render از state محلی خود بازیابی کنید و در صورت بروز ارور با استفاده از رندر شرطی ری‌اکت، یک پیام را نمایش دهید.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey,
error
} = this.state;
...
if (error) {
return <p>Something went wrong.</p>;
}
return (
<div className="page">

...
</div>
);
}
}
```

تمام شد. اگر می‌خواهید امتحان کنید که مدیریت خطای شما کار می‌کند، می‌توانید API URL را به چیز دیگری که وجود ندارد، تغییر دهید.

src/App.js

```
const PATH_BASE = 'https://hn.foo.bar.com/api/v1';
```

پس از آن، باید پیام خطا را به جای اپ خود دریافت کنید. این زمانی است که می‌خواهید رندر شرطی را برای نمایش پیغام خطا استفاده کنید. در این مورد، کل اپ دیگر نمایش داده نمی‌شود. این بهترین تجربه کاربری نیست. بنابراین، نمایش کامپوننت Table یا پیغام خطا چطور است؟ باقی‌مانده‌ی اپ در صورت ایجاد خطا هنوز قابل مشاهده خواهد بود.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
```

```
results,
searchKey,
error
} = this.state;
const page = (
results &&
results[searchKey] &&
results[searchKey].page
) || 0;
const list = (
results &&
results[searchKey] &&
results[searchKey].hits

) || [];
return (
<div className="page">
<div className="interactions">
...
</div>
{ error
? <div className="interactions">
<p>Something went wrong.</p>
</div>
: <Table
list={list}
onDismiss={this.onDismiss}
/>
}
...
</div>
);
}
}
```

در نهایت، فراموش نکنید که URL مربوط به API را به حالت اول برگردانید.

src/App.js

```
const PATH_BASE = 'https://hn.algolia.com/api/v1';
```

اپ شما هنوز هم باید کار کند، اما این بار با مدیریت خطاها، در صورتی که درخواست API نتواند انجام شود.

تمرین

- در مورد مدیریت ارورهای کامپوننت در ری‌اکت[۷۷] بیش‌تر بخوانید.

[77] https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html

Axios به‌جای دریافت

در یکی از گفتار های قبلی، شما از دریافت اولیه‌ی API برای اجرای یک درخواست از پلتفرم هکر نیوز استفاده کردید. مرورگر شما را قادر می‌سازد از دریافت اولیه‌ی API استفاده کنید. با این حال، تمام مرورگرها مخصوصاً مرورگرهای قدیمی از آن پشتیبانی نمی‌کنند.

علاوه بر این، هنگامی که شما شروع به تست اپ خود در یک محیط مرورگر headless می‌کنید (در آن هیچ مرورگری وجود ندارد، در عوض فقط تقلیدکننده است) ممکن است مشکلات مربوط به دریافت API وجود داشته باشد. یک محیط مرورگر headless مانند این می‌تواند هنگام نوشتن و اجرای آزمایشی برای اپ شما اتفاق بیافتد که در یک مرورگر واقعی اجرا نمی‌شود.

چند راه برای کار کردن fetch در مرورگرهای قدیمی (palyfills) و در تست (isomorphic- fetch[۷۸]) وجود دارد. اما در این کتاب وارد این دام نخواهیم شد.

یک راه جایگزین برای حل این مسأله، جایگزینی دریافت اولیه‌ی API با یک کتابخانه‌ی پایدار مانند Axios[۷۹] است. Axios کتابخآن‌های‌ست که تنها یک مشکل را حل می‌کند، اما آن را با کیفیت بالا حل می‌کند: انجام درخواست‌های غیرهمزمان در API های دور. به همین دلیل است که شما در این کتاب از آن استفاده خواهید کرد. در یک سطح عینی این گفتار باید به شما نشان دهد که چگونه می‌توانید یک کتابخانه را با یک کتابخانه‌ی دیگر جایگزین کنید. (که این‌جا یک API اولیه در مرورگر است). در سطح انتزاعی باید به شما نشان دهد که چطور همیشه می‌توانید راه حلی برای تغییرات ناگهانی (مثلاً مرورگرهای قدیمی، آزمایش‌های مرورگرهای Headlles) در توسعه وب پیدا کنید. بنابراین با وجود هر مانعی که سر راه شما باشد هرگز دست از جست‌وجوی راه حل‌های جدید برندارید.

پس بهتر است ببینیم چگونه دریافت اولیه‌ی API را می توان با axios جایگزین کرد. در واقع هرچه تا این‌جا گفته شده در عمل بسیار دشوارتر است. ابتدا باید axios را در خط فرمان نصب کنید:

Command Line

```
npm install axios
```

دوم ، باید axios را در فایل کامپوننت APP خود import کنید.

src/App.js

```
import React, { Component } from 'react';
import axios from 'axios';
import './App.css';
...
```

و آخرین مورد، شما می‌توانید آن را به‌جای ()fetch استفاده کنید.

[78] https://github.com/matthew-andrews/isomorphic-fetch
[79] https://github.com/axios/axios

استفاده از آن تقریباً مشابه با دریافت اولیه‌ی API است. URL را به عنوان آرگومان می گیرد و Promise را بر می‌گرداند. شما مجبور نیستید پاسخ را به JSON تغییر دهید. axios این کار را برای شما انجام می‌دهد و نتیجه را به یک شیء داده در جاوااسکریپت منتقل می‌کند. بنابراین اطمینان حاصل کنید که کد خود را با ساختار داده بازگشتی منطبق کنید.

src/App.js

```
class App extends Component {
...
fetchSearchTopStories(searchTerm, page = 0) {
axios(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}\
${page}&${PARAM_HPP}${DEFAULT_HPP}`)
.then(result => this.setSearchTopStories(result.data))
.catch(error => this.setState({ error }));
}
...
}
```

همین کار برای جایگزینی Fetch با axios در این گفتار کافی است. شما در کد خود axios را فراخوانی می‌کنید تا از درخواست پیش‌فرض HTTP GET آن استفاده کنید. با فراخوانی axios.get() شما می‌توانید درخواست GET را ایجاد کنید. هم‌چنین می‌توانید از یک متد HTTP دیگر مانند HTTP Post همراه با axios.Post() استفاده کنید. با فراخوانی axios get شما می‌توانید در خواست را ایجاد کنید. همچنین می‌توانید از یک متد HTTP دیگر مانند HTTP Post با axios.Post() استفاده کنید. در این‌جا می‌توانید ببینید که چگونه axios یک کتابخانه‌ی قدرتمند برای انجام درخواست در API راه دور است. معمولاً زمانی که درخواست API شما پیچیده می‌شود یا با تغییر ناگهانی توسط وب با promise ها سروکار دارید توصیه می‌کنم از این مقدمه بر API استفاده کنید. علاوه بر این، در گفتار بعد آزمایش اپ خود را انجام خواهید داد. پس از آن دیگر نیازی به نگرانی در مورد مرورگر یا محیط مرورگر headless نخواهید داشت.

می‌خواهم یک اصلاح دیگر برای درخواست هکر نیوز در کامپوننت APP را به شما معرفی کنم. کامپوننت خود را هنگامی که صفحه برای اولین بار در مرورگر نمایش داده می‌شود تصور کنید. در ComponentDidMount() کامپوننت شروع به ایجاد درخواست می‌کند. اما بعد از آن، چون اپ شما navigation دارد، از این صفحه به صفحه‌ای دیگر حرکت خواهید کرد. کامپوننت APP شما غیرفعال می‌شود. اما هنوز یک درخواست از متد چرخه‌ی زندگی ComponentDidMount() در انتظار است. این درخواست به مقصد استفاده از this.setState() و در نهایت در بلوک then یا catch در promise حرکت خواهد کرد. شاید پس از آن برای اولین بار شما هشدار زیر را در خط فرمان خود یا در خروجی توسعه‌دهنده‌ی مرورگر خود ببینید:

Command Line

```
Warning: Can only update a mounted or mounting component. This usually means you\
called setState, replaceState, or forceUpdate on an unmounted component. This i\
s a no-op
```

می‌توانید با حذف این درخواست هنگامی که کامپوننت شما غیر فعال است، با این مسأله برخورد کنید و یا از فراخوانی this.setState() در یک کامپوننت غیرفعال جلوگیری کنید. این بهترین عمل در ری‌اکت است، حتی اگر بسیاری از توسعه‌دهندگان از آن برای حفظ یک آپ تمیز بدون هیچ هشدار مزاحم پیروی نکنند. با این حال promise API جاری یک درخواست را لغو نمی‌کند. بنابراین شما باید در مورد این موضوع خودتان دست‌به‌کار شوید. ممکن است به همین دلیل باشد که بسیاری از توسعه‌دهندگان این بهترین روش را دنبال نمی‌کنند. فرمان زیر به نظر می‌رسد بیش‌تر شبیه به یک راه حل باشد تا بک اجرای پایدار. به همین دلیل می‌توانید انتخاب کنیدکه شما می‌خواهید به این دلیل که کامپوننت شما کار نمی‌کند روی هشدارها کار کنید یا نه. با این وجود، زمانی که این خطا در یکی از گفتارهای بعدی کتاب و یا در اپ شما اتفاق افتاد، این هشدار را در ذهن نگه دارید. آن هنگام می‌دانید چگونه با آن برخورد کنید.

بهتر است کار با آن را شروع کنیم. شما می‌توانید فیلد کلاسی را معرفی کنید که state چرخه‌ی زندگی را در کامپوننت شما نگه می‌دارد. می‌توان وقتی کامپوننت مقداردهی اولیه می‌شود، آن را به‌عنوان یک مقدار false آغاز کرد، و وقتی کامپوننت نصب شد، آن را به مقدار true تغییر داد، اما پس از آن که کامپوننت کار نکرد، دوباره آن را به false تغییر داد. به این ترتیب می‌توانید مسیر state چرخه‌ی زندگی کامپوننت خودتان را نگه دارید. این کار هیچ ارتباطی با state محلی ذخیره شده و اصلاح‌شده با this.state و this.setState() ندارد، زیرا باید بتوانید به‌طور مستقیم در کامپوننت، بدون دسترسی به مدیریت state محلی ری‌اکت، به آن دسترسی داشته باشید. علاوه بر این، این فرایند، زمانی که فیلد کلاس به این شیوه تغییر می‌کند، منجر به هیچ رندر مجددی در کامپوننت نمی‌شود.

src/App.js

```
class App extends Component {
_isMounted = false;
constructor(props) {
...
}
...
componentDidMount() {
this._isMounted = true;
const { searchTerm } = this.state;
this.setState({ searchKey: searchTerm });
this.fetchSearchTopStories(searchTerm);
}
componentWillUnmount() {
this._isMounted = false;
}
...
}
```

در نهایت، شما می‌توانید از این دانش استفاده کنید تا درخواست خود را لغو نکنید. اما برای اجتناب از this.setState() در کامپوننت خود، حتی اگر کامپوننت پیش از این نصب نشده باشد (یا کار نکرده باشد) این کار مانع از هشدار مذکور خواهد شد.

src/App.js

```
class App extends Component {
...
fetchSearchTopStories(searchTerm, page = 0) {
axios(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}\
${page}&${PARAM_HPP}${DEFAULT_HPP}`)
.then(result => this._isMounted && this.setSearchTopStories(result.data))
.catch(error => this._isMounted && this.setState({ error }));
}
...
}
```

در کل این گفتار به شما یاد داده شده است که چگونه می‌توانید یک کتابخانه را در ری‌اکت جایگزین یک کتابخانه‌ی دیگر کنید. اگر به هر مسأله‌ای برخوردید، می‌توانید برای کمک به خودتان از اکوسیستم کتابخانه‌ی وسیع جاوااسکریپت استفاده کنید. علاوه بر این، یک راه برای چگونگی اجتناب از فراخوانی this.setState() در ری‌اکت برای یک کامپوننت انجام‌نشده مشاهده کردید. اگر در کتابخانه‌ی axios عمیق شوید، یک راه برای جلوگیری از کنسل شدن درخواست در همان ابتدا پیدا می‌کنید. خواندن بیش‌تر در مورد این موضوع، به شما کمک می‌کند.

تمرین

- در مورد چرا فریم‌ورک‌ها مهم هستند[80]، بیش‌تر بخوانید.
- در مورد دیگر سینتکس های کامپوننت ری‌اکت[81] بیش‌تر بخوانید.

[80] https://www.robinwieruch.de/why-frameworks-matter/
[81] https://github.com/the-road-to-learn-react/react-alternative-class-component-syntax

شما آموختید که با یک API در ری‌اکت تعامل برقرار کنید. بیایید این گفتار را بازنویسی کنیم:

- **ری‌اکت:**
  - متدهای چرخه‌ی زندگی کامپوننت کلاس ES6 برای موارد متفاوت استفاده می شوند.
  - ()ComponentDidMount برای تعاملات شرطی
  - رندرینگ‌های شرطی
  - event های ترکیبی در فرم‌ها
  - مدیریت ارورها
  - قطع یک درخواست API از راه دور
- ES6 و فراتر از آن
  - رشته‌های قالب (template) برای نوشتن رشته‌ها
  - اپراتور گسترش برای ساختار داده‌ی تغییرناپذیر
  - محاسبه‌ی نام‌های ویژگی
  - فیلد کلاس
- عمومی
  - تعاملات API هکر نیوز
  - دریافت اولیه‌ی براوزر API
  - جست‌وجوی سمت سرور و سمت client
  - صفحه‌بندی داده‌ها
  - کش سمت مشتری
  - axios به عنوان یک جایگزین برای دریافت اولیه‌ی API

مجدداً احساس می کنم که یک استراحت داشته باشیم. آموزش‌ها را به‌طور کامل یاد بگیرید. می‌توانید با کدی که تا به حال نوشته اید، آزمایش‌های مختلف انجام دهید. شما می‌توانید کد سورس را در ریپازیتوری رسمی[۸۲] پیدا کنید.

[82] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.3.1

سازماندهی و تست کد

این گفتار بر روی موضوعات مهم تمرکز خواهد کرد تا بتواند کد شما را در یک اپ پایدار نگه دارد. در هنگام سازماندهی پوشه‌ها و فایل‌های خود، بهترین شیوه‌ها را در مورد سازماندهی کدها درک خواهید کرد. یکی دیگر از جنبه‌هایی که شما یاد خواهید گرفت، این است که تست کنید، که این کار برای قوی نگه داشتن کدهای شما مهم است. کل گفتار به یک گام عقب‌تر از اجرای اپ برگشته و چند مورد از این موضوعات را برای شما توضیح می‌دهد.

ماژول‌های import , export: ES6

در جاوااسکریپت ES6 شما می‌توانید ویژگی‌ها را از ماژول‌ها وارد و صادر کنید. این ویژگی‌ها می‌توانند توابع، کلاس‌ها، کامپوننت، ثابت‌ها و چیزهای دیگر باشند، اساساً همه‌ی چیزهایی که می‌توانید به یک متغیر اختصاص دهید. ماژول‌ها می‌توانند فایل‌های تعداد یا کل پوشه‌ها با یک فایل index به عنوان نقطه‌ی ورود باشند.

در ابتدای کتاب، پس از این که اپ خود را با create-react-app بوت‌استرپ کردید، چندین Export و Import را در فایل‌های اولیه‌ی خود داشته‌اید. اکنون زمان مناسبی برای توضیح آن‌هاست.

اظهارات Export و Import به شما کمک می‌کند تا کد را در فایل‌های مختلف به اشتراک بگذارید. پیش از این هم چندین راه حل برای این فرایند در جاوااسکریپت وجود داشت. این یک ضعف بود زیرا به جای داشتن چندین روش برای انجام یک عمل، ترجیح می‌دهید یک روش استاندارد را دنبال کنید. اکنون این یک رفتار طبیعی و اولیه از زمان بروز جاوااسکریپت ES6 است.

علاوه بر این، این اظهارات Export و Import تقسیم کد را پوشش می‌دهد. شما کد خود را در چند فایل توزیع می‌کنید تا بتوانید آن‌ها را قابل استفاده‌ی مجدد و قابل‌نگهداری نگه دارید. مورد اول درست است، چون می‌توانید یک قطعه کد را در چند فایل وارد کنید. مورد دوم هم درست است، زیرا شما تنها یک سورس دارید، که در آن قطعه‌ی کد خود را نگه می دارید.

آخرین مورد این که این اظهارات به شما کمک می‌کنند تا در مورد کپسوله‌سازی فکر کنید. نیاز نیست هر عملکردی از فایل Export شود. برخی از این ویژگی‌ها فقط باید در فایلی که در آن تعریف شده، استفاده شود. اسکپورت یک فایل اساساً API عمومی فایل است. فقط ویژگی‌های اکسپورت‌شده‌ی قابل استفاده‌ی مجدد در جای دیگری هستند. این بهترین تجزیه کپسوله‌سازی است.

اما اجازه دهید به بخش عملی کار برسیم. اظهارت Export و Import چطور کار می‌کند؟ مثال‌های زیر، از طریق به اشتراک گذاشتن یک یا چند متغیر در دو فایل، این اظهارات را نمایش می‌دهد. در نهایت، این رویکرد می‌تواند به فایل‌های چندگانه تعمیم پیدا کند و می‌تواند فراتر از متغیرهای ساده را به اشتراک بگذارد.

شما می‌توانید یک یا چند متغیر را صادر کنید. این عمل Export نامیده می‌شود.

Code Playground: file1.js

```
const firstname = 'robin';
const lastname = 'wieruch';
export { firstname, lastname };
```

و آن‌ها را در یک فایل دیگر با یک مسیردهی به فایل وارد کنید.

Code Playground: file2.js

```
import { firstname, lastname } from './file1.js';
console.log(firstname);
// output: robin
```

شما همچنین می‌توانید تمام متغیرهای Export صادر شده از فایل دیگر را به عنوان یک شیء Import کنید.

Code Playground: file2.js

```
import * as person from './file1.js';
console.log(person.firstname);
// output: robin
```

Import ها می‌توانند نام مستعار داشته باشند. این زمانی اتفاق می‌افتد که شما ویژگی‌ها را از چند فایل که با نام‌های مشابه Export شده اند، Import می‌کنید. به همین دلیل است که شما می‌توانید از یک نام مستعار استفاده کنید.

Code Playground: file2.js

```
import { firstname as foo } from './file1.js';
console.log(foo);
// output: robin
```

در پایان شما از بیان پیش‌فرض استفاده می‌کنید. این فرایند می‌تواند برای چند مورد استفاده شود:

- برای Export و Import یک قابلیت تنها
- برای برجسته کردن عملکرد اصلی از API صادر شده از یک ماژول
- برای داشتن قابلیت Import مجدد (fall back)

Code Playground: file1.js

```
const robin = {
firstname: 'robin',
lastname: 'wieruch',
};
export default robin;
```

شما می‌توانید برای Import کردن آکولاد را حذف کنید.

Code Playground: file2.js

```
import developer from './file1.js';
console.log(developer);
// output: { firstname: 'robin', lastname: 'wieruch' }
```

علاوه بر این، نام Import می‌تواند از نام default صادر شده، متفاوت باشد، شما همچنین می‌توانید از حرف ربط برای اتصال Export و import ها استفاده کنید.

Code Playground: file1.js

```
const firstname = 'robin';
const lastname = 'wieruch';
const person = {
firstname,
lastname,
};
export {
firstname,
lastname,
};
export default person;
```

Code Playground: file2.js

```
import developer, { firstname, lastname } from './file1.js';
console.log(developer);
// output: { firstname: 'robin', lastname: 'wieruch' }
console.log(firstname, lastname);
// output: robin wieruch
```

در Export شما می‌توانید خط‌های اضافی را حذف کنید و متغیرها را به صورت مستقیم Export کنید.

Code Playground: file1.js

```
export const firstname = 'robin';
export const lastname = 'wieruch';
```

این‌ها ویژگی‌های اصلی برای ماژول ES6 هستند. آن‌ها به شما کمک می‌کنند که کد خودتان را سازماندهی کنید، کدتان را قابل نگه‌داری کنید و API های ماژول قابل استفاده‌ی مجدد را طراحی کنید. همچنین می‌توانید کاربردهای Export و import ر ا تست کنید.

شما این کار را در یکی از گفتارهای بعد انجام خواهد داد.

تمرین

- در مورد import در ES6[۸۳] بیش‌تر بخوانید.
- در مورد Export در ES6[۸۴] بیش‌تر بخوانید.

[83] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
[84] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export

سازماندهی کد با ماژول ES6

شما ممکن است متعجب شوید: چرا ما بهترین شیوه‌ی تقسیم کد برای فایل src/APP.jS را دنبال نکردیم؟ در فایلی که اکنون در اختیار داریم کامپوننت‌های متعددی داریم که می‌تواند در فایل‌ها/ فولدرهای (ماژول‌های) خودشان تعریف شوند. به خاطر یادگیری ری‌اکت، همه‌ی آن‌ها در یک فایل نگه داشته‌ایم، اما هنگامی که اپ ری‌اکت شما رشد می‌کند، باید این کامپوننت‌ها را به چند بخش تقسیم کنید. فقط به این ترتیب است که اپ شما مطرح می‌شود.

در ادامه، چند ساختار ماژول را پیشنهاد می‌دهم که می‌توانید اعمال کنید. توصیه می‌کنم آن‌ها را به عنوان یک تمرین در پایان کتاب انجام دهید. برای حفظ سادگی کتاب، تقسیم کد را انجام نمی‌دهم و گفتارهای بعدی را با فایل src/APP.jS ادامه خواهم داد.

یک ساختار ماژول امکان‌پذیر به این شکل است.

Folder Structure

```
src/
index.js
index.css
App.js
App.test.js
App.css
Button.js
Button.test.js
Button.css
Table.js
Table.test.js
Table.css
Search.js
Search.test.js
Search.css
```

این ساختار، کامپوننت‌ها را در فایل‌های خودشان جدا می‌کند، اما به نظر نمی‌رسد خیلی امیدوارکننده باشد. می‌توانید چندین بار نام‌ها را تکرار کنید و فقط فایل‌های گسترش متفاوت باشند. ساختار ماژول دیگری می‌تواند به این شکل باشد:

Folder Structure

```
src/
index.js
index.css
App/
index.js
test.js
index.css
Button/
index.js
test.js
index.css
Table/
```

```
index.js
test.js
index.css
Search/
index.js
test.js
index.css
```

به نظر می‌رسد این کد تمیزتر از قبلی است. نام index در یک فایل نشان می‌دهد که این فایل نقطه‌ی ورود به فولدر است. این فقط یک قرارداد نام‌گذاری متداول است، اما شما می‌توانید از سیستم نام‌گذاری خودتان نیز استفاده کنید. در این ساختار ماژول، کامپوننت از طریق بیان کامپوننت در فایل جاوااسکریپت تعریف شده است، اما با تست و استایل هم همراه است.

گام دیگری که می‌تواند ثابت او کامپوننت‌ها را از یکدیگر جدا کند. این ثابت ها برای ساختن آدرس API URL هکر نیوز استفاده می شوند.

Folder Structure

```
src/
index.js
index.css
constants/
index.js
components/
App/
index.js
test.js
index..css
Button/
index.js
test.js
index..css
...
```

طبیعتاً ماژول‌ها باید در /src/constants و src/Components تقسیم شوند. اکنون فایل Src/constants/index می‌تواند مانند زیر باشد:

Code Playground: src/constants/index.js

```
export const DEFAULT_QUERY = 'redux';
export const DEFAULT_HPP = '100';
export const PATH_BASE = 'https://hn.algolia.com/api/v1';
export const PATH_SEARCH = '/search';
export const PARAM_SEARCH = 'query=';
export const PARAM_PAGE = 'page=';
export const PARAM_HPP = 'hitsPerPage=';
```

فایل APP/index .js می‌تواند این متغیرها را Export کند تا از آن‌ها استفاده کند.

Code Playground: src/components/App/index.js

```
import {
DEFAULT_QUERY,
DEFAULT_HPP,
PATH_BASE,
PATH_SEARCH,
PARAM_SEARCH,
PARAM_PAGE,
PARAM_HPP,
} from '../constants/index.js';
...
```

هنگامی که از نام‌گذاری index .js استفاده می‌کنید، می‌توانید نام فایل را در مسیردهی حذف کنید.

Code Playground: src/components/App/index.js

```
import {
DEFAULT_QUERY,
DEFAULT_HPP,
PATH_BASE,
PATH_SEARCH,
PARAM_SEARCH,
PARAM_PAGE,
PARAM_HPP,
} from '../constants';
...
```

اما منظور از نام‌گذاری index.js چیست؟ این مفهوم در قرارداد nade.js معرفی شد. فایل index قطعه‌ی ورود به ماژول است. این مسأله حضور API عمومی را در ماژول توصیف می‌کنید. ماژول‌های خارجی فقط مجاز به استفاده از فایل index.is برای import کدهای مشترک از ماژول هستند. برای نشان دادن آن، ساختار ماژول زیرا در نظر بگیرید:

Folder Structure

```
src/
index.js
App/
index.js
Buttons/
index.js
SubmitButton.js
SaveButton.js
CancelButton.js
```

فولدر /Buttons دارای چند کامپوننت مختلف است که در فایل‌های مجزا تعریف شده‌اند. هر فایل می‌تواند کامپوننت خاصی را ایجاد کند و به صورت پیش‌فرض صادر کند که از طریق مسیر Buttons/index.js قابل دسترسی باشد. فایل Buttons/index.js تمام این نمایه‌های مختلف را import می‌کند و آن‌ها را به عنوان ماژول API عمومی Export می‌کند.

Code Playground: src/Buttons/index.js

```
import SubmitButton from './SubmitButton';
import SaveButton from './SaveButton';
import CancelButton from './CancelButton';
export {
SubmitButton,
SaveButton,
CancelButton,
};
```

اکنون، src/APP/index.js می‌تواند button ها را از API ماژول عمومی واقع در فایل index.js وارد import کند.

Code Playground: src/App/index.js

```
import {
SubmitButton,
SaveButton,
CancelButton
} from '../Buttons';
```

با توجه به این، روش زیر نسبت به index.js در ماژول تمرین بدتری برای رسیدن به فایل‌های دیگر است. این تمرین قواعد کپسوله‌سازی را نقض می‌کند.

Code Playground: src/App/index.js

```
// bad practice, don't do it
import SubmitButton from '../Buttons/SubmitButton';
```

حالا شما می‌دانید چگونه می‌توانید با استفاده از محدودیت‌های کپسوله‌سازی، کد سورس خود را در ماژول‌ها نگه دارید. همانطور که گفتم، به خاطر ساده نگه داشتن این کتاب این تغییرات را اعمال نخواهم کرد. اما بعد از خواندن، باید خوتان بازسازی را انجام دهید.

تمرین:

- بعد از پایان کتاب، فایل src/APP.js خود را به چند کامپوننت ماژول تقسیم کنید.

تست Snapshot با Jest

این کتاب به‌طور عمقی به موضوع تست نخواهد پرداخت، اما تست‌ها نباید فراموش شوند. تست کد شما در برنامه‌نویسی ضروری است و باید به عنوان اجبار دیده شود. شما می‌خواهید کیفیت کد خود را بالا نگه دارید و مطمئن باشید که همه چیز درست کار می‌کند.

شاید شما در مورد هرم تست شنیده باشید. انواع تست‌های end-to-end، تست‌های یکپارچه و تست‌های واحد وجود دارند. اگر با آن‌ها آشنا نباشید، این کتاب به شما یک مرور سریع و ساده می‌دهد. یک تست واحد برای آزمایش یک بلوک مجرد و کوچک از کد استفاده می‌شود. این تست می‌تواند یک تابع تنها باشد که توسط تست واحد مورد آزمایش قرار می‌گیرد. با این حال، گاهی اوقات واحدها به‌تنهایی به خوبی کار می‌کنند، اما در ترکیب با واحدهای دیگر ممکن استعمل نکنند. آن‌ها باید به عنوان یک واحد گروهی مورد آزمایش قرار گیرند. این جایی‌ست که تست‌های یکپارچه می‌توانند با نشان دادن کارآیی یا عدم کارآیی واحد با یکدیگر، کمک کنند. آخرین مورد، یک تست end-to-end شبیه‌سازی یک سناریوی واقعی کاربر است. این تست می‌تواند یک راه‌اندازی خودکار در مرورگر باشد که شبیه‌سازی جریان ورود کاربر (login) به یک اپ وب است. در حالی که تست‌های واحد سریع و آسان برای نوشتن و نگهداری هستند، تست‌های end-to-end در انتهای دیگر محور سختی قرار می‌گیرند.

به چند تست برای هر نوع از واحدها نیاز داریم؟ شما می‌خواهید تست‌های واحد زیادی را برای پوشش دادن توابع خود داشته باشید. پس از آن، می‌توانید چند تست ادغام داشته باشید برای پوشش دادن این که توابع مهم در ترکیب، همان‌گونه که انتظار می‌رود، رفتار می‌کنند. در نهایت، ممکن است بخواهید فقط چند تست end-to-end برای شبیه‌سازی سناریوهای بحرانی در اپ وب خودتان داشته باشید. این برای یک سفر کلی در دنیای تست کردن کافی‌ست.

اکنون چگونه می‌توان این دانش را در تست اپ ری‌اکت خود اعمال کرد؟ پایه و اساس تست در ری‌اکت، تست کامپوننت‌هاست که می‌تواند به عنوان تست واحد و قسمتی از تست Snapshot تعمیم یابد. با استفاده از یک کتابخانه به نام «آنزیم» در گفتار بعدی تست واحد را برای کامپوننت‌هایتان انجام خواهید داد. در این گفتار شما بر روی نوع دیگری از تست تمرکز می‌کنید. این‌جاست که تست‌های Snapshot که وارد بازی می شوند.

Jest[۸۵] یک فریم‌ورک تست جاوااسکریپت است که در فیس‌بوک استفاده می‌شود. در جامعه‌ی ری‌اکت، این فریم‌ورک برای تست کامپوننت‌های ری‌اکت استفاده می‌شود. خوشبختانه create-react-app در حال حاضر با Jest همراه است، بنابراین لازم نیست در مورد تنظیم آن نگران باشید.

اکنون بهتر است شروع به تست اولین کامپوننت کنیم. قبل از این که بتوانید این کار را انجام دهید، باید کامپوننت‌هایتان را که می‌خواهید از فایل src/App.Js تست کنید، Export نمایید. پس از آن می‌توانید آن‌ها را در فایل دیگری آزمایش کنید. این مورد را در گفتار «سازماندهی کد» یاد گرفته اید.

src/App.js

```
...
class App extends Component {
...
}
...
```

[85] https://facebook.github.io/jest/

```
export default App;
export {
Button,
Search,
Table,
};
```

در فایل APP.test.Js شما اولین تست که با create-react-app ایجاد شده است را خواهید یافت. این نشان می‌دهد کامپوننت APP بدون هیچ گونه خطایی اجرا می‌شود.

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
it('renders without crashing', () => {
const div = document.createElement('div');
ReactDOM.render(<App />, div);
ReactDOM.unmountComponentAtNode(div);
});
```

بلوک "it" یک مورد آزمون را توصیف می‌کند. این بلوک با توضیحات تست همراه می‌شود و هنگامی که شما آن را تست می‌کنید، می‌تواند با موفقیت و یا شکست همراه شود. علاوه بر این، می‌توانید آن را در یک بلوک "Describe" قرار دهید که واحد تست شما را تعریف می‌کند. یک رشته‌ی آزمایشی می‌تواند شامل دسته ای از بلوک های "it" برای یک کامپوننت خاص باشد. در ادامه بلوک «توضیحات» را خواهید دید. هر دو بلوک برای جدا کردن و سازماندهی موارد تست شما مورد استفاده قرار می گیرند. توجه داشته باشید که تابع در جامعه‌ی جاوااسکریپت به عنوان تابعی که در آن شما یک تست یگانه را اجرا می‌کنید، تأیید می‌شود. با این حال، در Jest اغلب به عنوان یک تابع تست مستعار (alias) یافت می‌شود.

می‌توانید نمونه‌های تست خود را با استفاده از اسکرپیت تست تعاملی create-react-app در خط فرمان اجرا کنید. خروجی را برای تمام موارد تست، در خط فرمان دریافت خواهید کرد.

Command Line

```
npm test
```

اکنون jest شما را قادر به نوشتن تست Snapshot می‌کند. این تست یک Snapshot از کامپوننت‌های رندر شده‌ی شما ایجاد می‌کند و این Snapshot را در برابر Snapshot های آینده اجرا می‌کند. هنگامی که Snapshot آینده تغییر می‌کند در تست متوجه خواهید شد. شما می‌توانید تغییر Snapshot را قبول کنید، زیرا پیاده‌سازی کامپوننت را به صورت هدفمند تغییر داده‌اید، یا این تغییر را رد کرده و ارور را بررسی می‌کنید. تست واحد بسیار خوب است، زیرا فقط تفاوت خروجی رندر

شده را تست می‌کنید. این تست هزینه‌های نگهداری زیادی را اضافه نمی‌کند، زیرا هنگامی که شما چیزی را به صورت هدفمند برای خروجی رندر شده در کامپوننت تغییر داده باشید، می‌توانید به سادگی تغییرات Snapshot را بپذیرید.

jest , Snapshot را در یک فولدر ذخیره می‌کند. فقط از این راه می‌تواند نظارت در یک Snapshot در آینده را تأیید کند. علاوه بر این، Snapshot ها می‌توانند با نگه داشته شدن در یک پوشه، بین تیم‌ها به اشتراک گذاشته شوند.

قبل از نوشتن اولین تست Snapshot با استفاده از jest، باید یک کتابخانه‌ی مفید را نصب کنید.

Command Line

```
npm install --save-dev react-test-renderer
```

اکنون می‌توانید با استفاده از اولین تست Snapshot خود، تست کامپوننت APP گسترش دهید. ابتدا قابلیت‌های جدید را از پکیج نود import کنید و بلوک قبلی "it" را در یک بلوک "describe" بگذارید. در این حالت، مجموعه‌ی تست فقط برای کامپوننت APP است.

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import renderer from 'react-test-renderer';
import App from './App';
describe('App', () => {
it('renders without crashing', () => {
const div = document.createElement('div');
ReactDOM.render(<App />, div);
ReactDOM.unmountComponentAtNode(div);
});
});
```

اکنون می‌توانید اولین تست Snapshot خود را با استفاده از یک بلوک "test" اجرا کنید.

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import renderer from 'react-test-renderer';
import App from './App';
describe('App', () => {
it('renders without crashing', () => {
const div = document.createElement('div');
ReactDOM.render(<App />, div);
ReactDOM.unmountComponentAtNode(div);
```

```
});
test('has a valid snapshot', () => {
const component = renderer.create(
<App />
);
let tree = component.toJSON();
expect(tree).toMatchSnapshot();
});
});
```

دوباره تست‌های خود را اجرا کنید و ببینید آزمایش‌ها موفق خواهند بود یا شکست می‌خورند. آن‌ها باید موفق باشند. هنگامی که خروجی بلوک رندر را در کامپوننت APP خود تغییر می دهید، تست Snapshot باید شکست بخورد. سپس می‌توانید تصمیم بگیرید که Snapshot را به‌روزرسانی کنید یا کامپوننت APP خود را بررسی کنید.

اساساً تابع ()renderer.create یک Snapshot از کامپوننت APP شما ایجاد می‌کند. آن را به‌طور مجازی رندر می‌کند و DOM را در یک Snapshot ذخیره می‌نماید. پس از آن انتظار می رود که Snapshot با Snapshot قبلی که دفعه‌ی پیش از تست Snapshot داشته‌اید، مطابقت داشته باشد. به این ترتیب، می‌توانید اطمینان حاصل کنید که DOM باقی می‌ماند و با تصادفاً تغییر نمی‌کند.

اجازه دهید تست‌های بیش‌تری به کامپوننت مستقل‌مان اضافه کنیم. اول کامپوننت Search :

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import renderer from 'react-test-renderer';
import App, { Search } from './App';
...
describe('Search', () => {
it('renders without crashing', () => {
const div = document.createElement('div');
ReactDOM.render(<Search>Search</Search>, div);
ReactDOM.unmountComponentAtNode(div);
});
test('has a valid snapshot', () => {
const component = renderer.create(
<Search>Search</Search>
);
let tree = component.toJSON();
expect(tree).toMatchSnapshot();
});
});
```

کامپوننت Search دو تست مشابه به کامپوننت APP دارد. تست اول به‌راحتی کامپوننت Search را در DOM رندر می‌کند و تأیید می‌کند که در طول فرآیند رندر هیچ خطایی وجود ندارد. اگر یک خطا وجود داشته باشد، تست قطع خواهد شد، حتی اگر هیچ اعلانی (مثلاً expect , Match یا equel) در بلوک تست وجود نداشته باشد. تست Export دوم برای ذخیره‌ی یک Export از کامپوننت رندر شده و برای اجرای آن در برابر Export قبلی استفاده می‌شود. این تست زمانی که Snapshot تغییر کرده باشد، fail می‌شود.

دوم، شما می‌توانید کامپوننت Button را تست کنید، در حالی که تست‌های مشابه کامپوننت search اعمال می شوند.

src/App.test.js

```
...
import App, { Search, Button } from './App';
...
describe('Button', () => {
it('renders without crashing', () => {
const div = document.createElement('div');
ReactDOM.render(<Button>Give Me More</Button>, div);
ReactDOM.unmountComponentAtNode(div);
});
test('has a valid snapshot', () => {
const component = renderer.create(
<Button>Give Me More</Button>
);
let tree = component.toJSON();
expect(tree).toMatchSnapshot();
});
});
```

در نهایت، کامپوننت Table که شما می‌توانید یک دسته از props اولیه را به آن انتقال دهید تا آن را با یک لیست نمونه رندر کنید.

src/App.test.js

```
...
import App, { Search, Button, Table } from './App';
...
describe('Table', () => {
const props = {
list: [
{ title: '1', author: '1', num_comments: 1, points: 2, objectID: 'y' },
{ title: '2', author: '2', num_comments: 1, points: 2, objectID: 'z' },
],
};
```



```
it('renders without crashing', () => {
```

```
const div = document.createElement('div');
ReactDOM.render(<Table { ...props } />, div);
});
test('has a valid snapshot', () => {
const component = renderer.create(
<Table { ...props } />
);
let tree = component.toJSON();
expect(tree).toMatchSnapshot();
});
});
```

تست‌های Snapshot معمولاً بسیار پایه‌ای هستند. شما فقط می‌خواهید مطمئن شوید که خروجی کامپوننت تغییر نکند. زمانی که کامپوننت خروجی را تغییر دهد، باید تصمیم بگیرید که تغییرات را بپذیرید. در غیر این صورت زمانی که خروجی، همان خروجی مورد نظر شما نیست، باید کامپوننت را تعمیر کنید.

تمرین

- ببینید چطور تست Snapshot وقتی مقدار برگشتی کامپوننت را در متد رندر تغییر می دهید، fail می‌شود.
  - تغییرات Snapshot را یا بپذیرید یا رد کنید.
- هنگامی که کامپوننت‌ها در گفتارهای بعدی تغییر می‌کنند، تست‌های Snapshot خود را به‌روز نگه دارید.
- در مورد Jest در ری‌اکت[۸۶] بیش‌تر بخوانید.

[86] https://jestjs.io/docs/en/tutorial-react

تست‌های واحد با Enzyme

آنزیم[87] یک ابزار تست‌شده توسط Airbnb برای اثبات، دستکاری و تغییر کامپوننت‌های ری‌اکت است. شما می‌توانید از آن برای انجام تست‌های واحد برای تکمیل Snapshot خود در ری‌اکت استفاده کنید.

بیایید ببینیم چگونه می‌توانید از آنزیم استفاده کنید. ابتدا باید آن را نصب کنید، زیرا از طریق create-react-APP به صورت پیش‌فرض نمی آید. همچنین می‌توانید با extension از آن دری‌اکت استفاده کنید.

Command Line

```
npm install --save-dev enzyme react-addons-test-utils enzyme-adapter-react-16
```

دوم، شما باید آن را در تنظیمات تست خود وارد کنید و آن را برای استفاده در ری‌اکت مقداردهی اولیه نمایید.

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import renderer from 'react-test-renderer';
import Enzyme from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import App, { Search, Button, Table } from './App';
Enzyme.configure({ adapter: new Adapter() });
```

حالا شما می‌توانید اولین تست واحد خود را در Table بلوک "describe" بنویسید. شما از Shallow() برای رندر کامپوننت خودتان استفاده می‌کنید و تأکید می‌کنید که Table دارای دو آیتم است، چون شما دو آیتم لیست را انتقال می‌دهید. این تأکید به سادگی بررسی می‌کند تا مطمئن شود این المنت خود دو المنت با کلاس table-raw داشته باشد.

src/App.test.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import renderer from 'react-test-renderer';
import Enzyme, { shallow } from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';
import App, { Search, Button, Table } from './App';
...
describe('Table', () => {
const props = {
list: [
```

[87] https://github.com/airbnb/enzyme

```
{ title: '1', author: '1', num_comments: 1, points: 2, objectID: 'y' },
{ title: '2', author: '2', num_comments: 1, points: 2, objectID: 'z' },
],
};
...
it('shows two items in list', () => {
const element = shallow(
<Table { ...props } />
);
expect(element.find('.table-row').length).toBe(2);
});
});
```

Shallow کامپونت را بدون کامپونت‌های فرزند رندر می‌کند. به این ترتیب، می‌توانید تست را خیلی اختصاصی برای یک کامپونت انجام دهید.

آنزیم به صورت کلی دارای سه مکانیسم رندر در API است. شما از قبل Shallow() را می‌شناسید. اما Mount() و render() هم وجود دارند. هر دوی آن‌ها کامپونت والد و تمام کامپونت‌های فرزند را معرفی می‌کنند. علاوه بر این mount() به شما دسترسی به متدهای چرخه‌ی زندگی کامپونت را هم می‌دهد. اما چه زمانی از مکانیزم رندر استفاده کنیم؟ در اینجا برخی از قوانی ذکر شده‌اند:

- همیشه با تست Shallow شروع کنید.
- ComponentDidMount() یا ComponentDidUpdate() باید با استفاده از mount() مورد تست قرار گیرند.
- اگر می‌خواهید چرخه‌های زندگی کامپونت و رفتار فررزندان را تست کنید از mount() استفاده کنید.
- اگر می‌خواهید خروجی فرزندان کامپونت با کدهایی کمتر از mount() تست شوند و علاقه‌مند به متدهای چرخه‌ی زندگی نیستید، از render() استفاده کنید.

می‌توانید به تست واحد کامپونت‌های خود ادامه دهید. اما مطمئن شوید که تست‌ها ساده و قابل به‌روزرسانی هستند. در غیر این صورت مجبور خواهید شد که آن‌ها پس از تغییر کامپونت خود تغییر دهید. به همین دلیل است که فیس‌بوک همان ابتدا تست‌های snapshot خود را jest معرفی کرد.

تمرین

- برای کامپونت Button خود یک تست واحد با آنزیم بنویسید.
- در گفتارهای بعد تست واحد خود را به‌روز نگه دارید.
- در مورد آنزیم و رندرینگ API آن[88] بیش‌تر بخوانید.

[88] https://github.com/airbnb/enzyme

رابط کامپوننت با PropTypes

شما ممکن است TypeScript[89] یا Flow[90] را بشناسید تا نوع رابط برای جاوااسکریپت را معرفی کنید. یک زبان Typed دارای خطای کمتر است، زیرا کد بر اساس متن برنامه تأیید شده است. Editor ها و سایر سرویس‌ها می‌توانند قبل از اجرای برنامه این خطاها را دریافت کنند. این کار برنامه‌ی شما را قوی‌تر می‌کند.

این کتاب Flow یا TypeScript را معرفی نمی‌کند، اما یک روش بهتر برای بررسی Type ها در کامپوننت وجود دارد. برای جلوگیری از باگ‌ها، یک type checker درونی در ری‌اکت قرار داده شده است. شما می‌توانید از prop types برای توصیف رابط کاربری خود استفاده کنید. تمام props هایی که از یک کامپوننت والد به یک کامپوننت فرزند منتقل می‌شود، بر اساس رابط کاربری prop types که به کامپوننت فرزند اختصاص داده شده، اعتبار سنجی می‌شود. این گفتار به شما یاد می‌دهد که چگونه می‌توانید type کامپوننت‌های خود را با prop types ایمن کنید. من تغییرات را تا گفتارهای بعدی حذف خواهم کرد، زیرا آن‌ها کدهای غیرضروری را اضافه می کنند. اما شما باید آن‌ها را در طول مسیر نگه دارید و آن‌ها را به‌روز نگه دارید تا مطمئن شوید types رابط کاربری کامپوننت شما امن است.

ابتدا شما باید یک پکیج مجزا برای ری‌اکت نصب کنید.

Command Line

```
npm install prop-types
```

حالا می‌توانید prop types ها را import کنید.

src/App.js

```
import React, { Component } from 'react';
import axios from 'axios';
import PropTypes from 'prop-types';
```

اکنون شروع کنیم به اختصاص دادن رابط کاربری prop برای کامپوننت‌ها:

src/App.js

```
const Button = ({
onClick,
className = '',
children,
}) =>
```

[89] https://www.typescriptlang.org
[90] https://flow.org

```
<button
onClick={onClick}
className={className}
type="button"
>
{children}
</button>
Button.propTypes = {
onClick: PropTypes.func,
className: PropTypes.string,
children: PropTypes.node,
};
```

اساساً این تمام فرایند است. شما هر آرگومان را از امضای تابع می‌گیرید و prop types به آن اختصاص می‌دهید. prop types های پایه برای اشیاء ساده و پیچیده به عبارت زیر هستند:

- prop types.array
- prop types.bool
- prop types.func
- prop types.number
- prop types .object
- prop types.string

علاوه بر این‌ها شما دو prop types دیگر برای تعریف یک قطعه قابل رندر (node) دارید، به عنوان مثال، یک رشته و یک المنت ری‌اکت.

- prop types.node
- prop types.element

شما قبلاً از نود propTypes برای کامپوننت Button استفاده کرده‌اید. در کل تعاریف Export بیش‌تری وجود دارد که می‌توانید در اسناد رسمی ری‌اکت بخوانید.

در حال حاضر، تمام propTypes های تعریف‌شده برای Button اختیاری است. پارامترها می‌توانند Null یا undefined باشند. اما برای چندین props باید آن‌ها را به صورت defined تعریف کنید. شما می‌توانید این نیازمندی‌ها را برای این props ها به کامپوننت انتقال دهید.

src/App.js

```
Button.propTypes = {
onClick: PropTypes.func.isRequired,
className: PropTypes.string,
```

```
children: PropTypes.node.isRequired,
};
```

Class name ضروری نیست، زیرا می‌تواند به صورت پیش‌فرض یک رشته‌ی خالی باشد. پس از آن، شما باید یک رابط prop types برای کامپوننت Table تعریف کنید.

src/App.js

```
Table.propTypes = {
list: PropTypes.array.isRequired,
onDismiss: PropTypes.func.isRequired,
};
```

می‌توانید محتوای prop types آرایه را به صراحت تعریف کنید.

src/App.js

```
Table.propTypes = {
list: PropTypes.arrayOf(
PropTypes.shape({
objectID: PropTypes.string.isRequired,
author: PropTypes.string,
url: PropTypes.string,
num_comments: PropTypes.number,
points: PropTypes.number,
})
).isRequired,
onDismiss: PropTypes.func.isRequired,
};
```

فقط Object.ID ضروری است، زیرا می‌دانید که قسمتی از کد شما به آن بستگی دارد. ویژگی‌های دیگر فقط نمایش داده می شوند. بنابراین لزوماً ضروری نیستند. علاوه بر این، شما نمی‌توانید مطمئن باشید که API هکر نیوز همیشه یک ویژگی برای هر شیء در آرایه تعریف می‌کند.

بخش prop types تمام شد. اما یک جنبه‌ی دیگر وجود دارد. در کامپوننت خود می‌توانید propهای پیش‌فرض تعریف کنید. بیایید دوباره به کامپوننت Button بپردازیم. ویژگی Class name یک پارامتر پیش‌فرض ES6 در توزیع کامپوننت دارد.

src/App.js

```
const Button = ({
onClick,
className = '',
children
}) =>
...
```

می‌توانید آن را با prop پیش‌فرض داخلی ری‌اکت جایگزین کنید.

src/App.js

```
const Button = ({
onClick,
className,
children
}) =>
<button
onClick={onClick}
className={className}
type="button"
>
{children}
</button>
Button.defaultProps = {
className: '',
};
```

همانند پارامتر پیش‌فرض ES6، زمانی که کامپوننت والد مقدار پیش‌فرض را تعیین نکرده باشد، ویژگی پیش‌فرض اطمینان حاصل می‌کند که ویژگی به عنوان مقدار پیش‌فرض تعیین شده باشد. بررسی نوع prop.Types بعد از مقداردهی ویژگی پیش‌فرض اتفاق می‌افتد.

اگر تست‌های خود را دوباره اجرا کنید، ممکن است اورهای prop types را برای کامپوننت‌های خود، در خط فرمان مشاهده کنید. این اتفاق می‌تواند به این دلیل بیافتد که تمام propهایی که در تعریف prop types ضروری هستند را در کامپوننت برای تست، تعریف نکرده‌اید. هرچند تست‌ها همه درست انجام می شوند. برای اجتناب از این اورها می‌توانید تمام propهایی که ضروری هستند را در تست‌هایتان به کامپوننت انتقال دهید.

تمرین:

- رابط کاربری prop types را برای کامپوننت Search تعریف کنید.
- رابط کاربری prop types را هنگام اضافه کردن و به‌روزرسانی کامپوننت در گفتار های بعدی اضافه و به‌روزرسانی کنید.

- درباره propTypes ری‌اکت[۹۱] بیش‌تر بخوانید.

[91] https://reactjs.org/docs/typechecking-with-proptypes.html

در این گفتار یاد گرفته‌اید که چگونه کد خود را سازماندهی کنید و چگونه آن را آزمایش کنید. بیایید گفتار را مرور کنیم:

- **ری‌اکت:**
  - prop types اجازه می‌دهد بررسی types را برای کامپوننت تعریف کنید.
  - Jest اجازه می‌دهد تا تست‌های prop types را برای کامپوننت‌هایتان بنویسید.
  - آنزیم اجازه می‌دهد تا تست‌های واحد را برای کامپوننت‌هایتان بنویسید.
- **ES :**
  - import و Export به شما کمک می‌کند تا کد خود را سازماندهی کنید.
- **عمومی:**
  - سازماندهی به شما امکان می‌دهد تا اپ خود را به بهترین شکل مقداردهی کنید.

شما می‌توانید سورس کد را در ریپازیتوری رسمی[۹۲] پیدا کنید.

[92] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.4

کامپوننت‌های پیشرفته‌ی ری‌اکت

این گفتار بر روی اجرای کامپوننت‌های پیشرفته‌ی ری‌اکت تمرکز دارد. شما در مورد کامپوننت‌های مرتبه‌ی بالاتر و نحوه‌ی اجرای آن‌ها خواهید آموخت. علاوه بر این، به موضوعات پیشرفته‌ای در ری‌اکت و تعاملات پیچیده با آن می پردازید.

Ref در المنت DOM

گاهی اوقات شما نیاز دارید با نودهای DOM در ری‌اکت تعامل داشته باشید. ویژگی ref می‌تواند به یک نود در المنت شما دسترسی پیدا کند. معمولاً این یک ضد الگوی در ری‌اکت است، زیرا باید از شیوه‌ی اعلام رسمی و جریان داده یک‌طرفه‌ی آن استفاده کنید. هنگامی که اولین فیلد ورودی جست‌وجو را تعریف کردید، درباره‌ی آن یاد گرفتید. اما موارد خاصی وجود دارد که در آن نیاز به دسترسی به نودهای DOM دارید. اسناد رسمی سه مورد استفاده را ذکر می‌کند:

- برای استفاده از DOM API (تمرکز، بخش رسانه‌ای و غیره)
- برای فراخوانی ضروری انیمیشن‌های نودهای DOM
- برای ادغام با یک کتابخانه‌ی شخص ثالث که نیاز به نودهای DOM دارد. (به عنوان مثال D3.js[۹۳])

بیایید این تعامل را با مثال کامپوننت Search انجام دهیم. هنگامی که اپ برای اولین بار رندر می‌شود، فیلد ورودی باید متمرکز شود. این یک مورد استفاده است که شما نیاز به دسترسی به DOM API دارید. این گفتار به شما نشان می‌دهد که این کار چگونه انجام می‌شود، اما از آن‌جایی که خیلی برای خود اپ مفید نیست، تغییرات را در انتهای گفتار حذف خواهیم کرد. هرچند شما می‌توانید آن را در اپ خود نگه دارید.

به طور کلی، می‌توانید ویژگی ref را در هر دو کامپوننت بدون state عملی و کلاس ES6 استفاده کنید. در مثال استفاده از تمرکز، شما به یک متد چرخه‌ی زندگی نیاز دارید. به همین دلیل این رویکرد با استفاده از ویژگی ref در کامپوننت کلاس ES6 نمایش داده می‌شود.

گام اولیه این است که کامپوننت بدون state را به یک کامپوننت کلاس ES6 بازنویسی کنید.

src/App.js

```
class Search extends Component {
render() {
const {
value,
onChange,
onSubmit,
children
} = this.props;
return (
<form onSubmit={onSubmit}>
<input
type="text"
value={value}
onChange={onChange}
/>
<button type="submit">
{children}
</button>
</form>
```

[93] https://d3js.org

```
);
}
}
```

شیء this در کامپوننت کلاس Export به ما کمک می‌کند تا بتوانیم به نودهای DOM با ویژگی ref مراجعه کنیم.

src/App.js

```
class Search extends Component {
render() {
const {
value,
onChange,
onSubmit,
children
} = this.props;
return (
<form onSubmit={onSubmit}>
<input
type="text"
value={value}
onChange={onChange}
ref={(node) => { this.input = node; }}
/>
<button type="submit">
{children}
</button>
</form>
);
}
}
```

اکنون، هنگامی که کامپوننت با استفاده از شیء this، متد چرخه‌ی زندگی مناسب و Export DOM API، mount شده است، می‌توانید بر روی فیلتر ورودی متمرکز شوید.

src/App.js

```
class Search extends Component {
componentDidMount() {
if(this.input) {
this.input.focus();
}
}
render() {
const {
```

```
value,
onChange,
onSubmit,
children
} = this.props;
return (
<form onSubmit={onSubmit}>
<input
type="text"
value={value}
onChange={onChange}
ref={(node) => { this.input = node; }}
/>
<button type="submit">
{children}
</button>
</form>
);
}
}
```

فیلد ورودی باید هنگامی که اپ رندر می‌شود، متمرکز شده باشد. این اساساً برای استفاده از ویژگی ref است.

اما چطور می‌توانید به این ویژگی ref در تابع کامپوننت بدون state دسترسی پیدا کنید. بدون استفاده از شیء this تابع زیر را کامپوننت بدون state را نشان می‌دهد.

src/App.js

```
const Search = ({
value,
onChange,
onSubmit,
children
}) => {
let input;
return (
<form onSubmit={onSubmit}>
<input
type="text"
value={value}
onChange={onChange}
ref={(node) => input = node}
/>
<button type="submit">
{children}
</button>
</form>
```

```
);
}
```

اکنون می‌توانید به المنت DOM ورودی دسترسی داشته باشید. در مثال استفاده از تمرکز، این کار به شما کمک نمی‌کند، زیرا در یک کامپوننت بدون state کاربردی هیچ متد چرخه‌ی زندگی برای ایجاد تمرکز ندارید. اما در آینده ممکن است در شرایطی قرار بگیرید که استفاده از ویژگی ref در یک کامپوننت بدون state کاربردی، منطقی باشد.

تمرین:

- در مورد استفاده از ویژگی ref در ری‌اکت[94] بیش‌تر بخوانید.
- در مورد ویژگی ref به طور کلی در ری‌اکت[95] بیش‌تر بخوانید.

[94] https://www.robinwieruch.de/react-ref-attribute-dom-node/
[95] https://reactjs.org/docs/refs-and-the-dom.html

در حال بارگذاری...

بیایید به اپ برگردیم. ممکن است بخواهید هنگامی که یک درخواست جست‌وجو به API هکر نیوز ارسال می‌کنید، شاخص loading را نشان دهید. این درخواست ناهمزمان است و مجبور هستید بازخوردهای خود را در مورد آن‌چه که اتفاق می افتد، به کاربر نشان دهید. اجازه دهید کامپوننت loading قابل استفاده را در فایل src/APP-js تعریف کنیم.

src/App.js

```
const Loading = () =>
<div>Loading ...</div>
```

اکنون به یک ویژگی برای ذخیره‌ی state در حال بارگذاری نیاز دارید. براساس state بارگذاری، شما باید تصمیم بگیرید چه زمان‌هایی کامپوننت loading را نمایش دهید.

src/App.js

```
class App extends Component {
_isMounted = false;
constructor(props) {
super(props);
this.state = {
results: null,
searchKey: '',
searchTerm: DEFAULT_QUERY,
error: null,
isLoading: false,
};
...
}
...
}
```

مقدار اولیه‌ی ویژگی isLoading برابر با false است. قبل از این که کامپوننت APP نصب شود هیچ چیزی را لود نمی‌کنید.

وقتی درخواست را انجام می‌دهید، وضعیت بارگذاری را به true تغییر می‌دهید. در نهایت درخواست با موفقیت انجام خواهد شد و شما می‌توانید state loading را به false تنظیم کنید.

src/App.js

```
class App extends Component {
...
setSearchTopStories(result) {
...
this.setState({
```

```
results: {
...results,
[searchKey]: { hits: updatedHits, page }
},
isLoading: false
});
}
fetchSearchTopStories(searchTerm, page = 0) {
this.setState({ isLoading: true });
axios(`${PATH_BASE}${PATH_SEARCH}?${PARAM_SEARCH}${searchTerm}&${PARAM_PAGE}\
${page}&${PARAM_HPP}${DEFAULT_HPP}`)
.then(result => this._isMounted && this.setSearchTopStories(result.data))
.catch(error => this._isMounted && this.setState({ error }));
}
...
}
```

در آخرین مرحله، شما از کامپوننت loading در اپ خود استفاده خواهید کرد. یک رندر شرطی، بر اساس state لودینگ تصمیم می‌گیرد که آیا کامپوننت loading را نمایش دهد یا کامپوننت Button را. مورد اخیر دگمه‌ای برای دریافت داده‌های بیش‌تر است.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey,
error,
isLoading
} = this.state;
...
return (
<div className="page">
...
<div className="interactions">
{ isLoading
? <Loading />
: <Button
onClick={() => this.fetchSearchTopStories(searchKey, page + 1)}>
More
</Button>
}
</div>
</div>
);
```

```
}
}
```

در ابتدا کامپوننت loading نشان می‌دهد شما چه زمانی اپ خود را استارت می‌زنید، زیرا در ComponentDidMount() یک درخواست ایجاد کرده‌اید. در این‌جا کامپوننت Table وجود ندارد زیرا لیست خالی است. هنگامی که پاسخ از API هکر نیوز بازگردانده می‌شود، نتیجه نشان داده می‌شود و loading به Export تنظیم شده و کامپوننت loading ناپدید می‌شود. در عوض دگمه‌ی More برای گرفتن داده‌های بیش‌تر ظاهر می‌شود. هنگامی که داده‌های بیش‌تری را می‌گیرید، دگمه دوباره ناپدید شده و کامپوننت loading ظاهر خواهد شد.

تمرین:

- برای نمایش آیکون لودینگ به جای متن "loading" از یک کتابخانه مانند Font Awesome[۹۶] استفاده کنید.

[96] http://fontawesome.io

کامپوننت‌های مرتبه‌ی بالاتر

کامپوننت‌های مرتبه‌ی بالاتر (HOC) یک مفهوم پیشرفته در ری‌اکت هستند. HOC برابر با توابع مرتبه‌ی بالاتر است. آن‌ها هر نوع ورودی – اغلب اوقات کامپوننت‌ها، اما گاهی آرگومان‌ها – را می‌گیرند و یک کامپوننت را به عنوان خروجی باز می‌گردانند. کامپوننت بازگشتی، نسخه‌ی پیشرفته‌ی کامپوننت ورودی است و می‌تواند در JSX شما استفاده شود.

HOC ها برای موارد مختلف استفاده می‌شوند. آن‌ها می‌توانند ویژگی‌های را آماده کنند، state را مدیریت کنند یا نمایش یک کامپوننت را تغییر دهند. یک مورد استفاده می‌تواند به‌کارگیری HOC به عنوان یک helper برای یک رندر شرطی باشد. تصور کنید شما یک کامپوننت List دارید که یک لیست از آیتم‌ها یا هیچ را رندر می‌کند، چون لیست خالی یا null است. HOC می‌تواند به عنوان سپر استفاده شود که وقتی هیچ لیستی وجود ندارد، لیست هیچ چیز را رندر کند. از سوی دیگر کامپوننت List ساده نیاز به هیچ نگرانی در مورد موجود نبودن لیست ندارد. این کامپوننت فقط به رندر کردن لیست توجه دارد.

اجازه دهید یک HOC ساده انجام دهیم که کامپوننت را به عنوان ورودی می‌گیرد و یک کامپوننت باز می‌گرداند. شما می‌توانید آن را در فایل src/APP.js خود نگه دارید.

src/App.js

```
function withFoo(Component) {
return function(props) {
return <Component { ...props } />;
}
}
```

یک قرارداد تمیز برای نام‌گذاری HOC با پیشوند with وجود دارد. از آن‌جا که شما از جاوااسکریپت ES6 استفاده می‌کنید، می‌توانید HOC را به طور خلاصه با تابع arrow ES6 بیان کنید.

src/App.js

```
const withFoo = (Component) => (props) =>
<Component { ...props } />
```

در این مثال، کامپوننت ورودی، به عنوان همان کامپوننت خروجی باقی خواهد ماند. هیچ اتفاقی نمی‌افتد. این کد همان کامپوننت نمونه را رندر می‌کند و تمام props ها را به کامپوننت خروجی انتقال می‌دهد. اما این کار بی‌فایده است. اجازه دهید کامپوننت خروجی را گسترش دهیم. کامپوننت خروجی باید زمانی که state لودینگ برابر با true باشد، کامپوننت loading را نشان دهد، در غیر این صورت باید کامپوننت ورودی را نمایش دهد. رندر شرطی یک مورد استفاده‌ی بزرگ برای HOC است.

src/App.js

```
const withLoading = (Component) => (props) =>
props.isLoading
```

```
? <Loading />
: <Component { ...props } />
```

بر اساس ویژگی لودینگ شما می‌توانید یک رندر شرطی را اعمال کنید. این تابع کامپوننت loading یا کامپوننت ورودی را باز خواهد گرداند.

به طور کلی این ویژگی می‌تواند برای گسترش یک شیء بسیار مؤثر باشد. مانند شیء props در مثال قبلی به عنوان ورودی برای یک کامپوننت، تفاوت را در کد زیر ببینید.

Code Playground

```
// before you would have to destructure the props before passing them
const { foo, bar } = props;
<SomeComponent foo={foo} bar={bar} />
// but you can use the object spread operator to pass all object properties
<SomeComponent { ...props } />
```

یک مورد کوچک وجود دارد که باید از آن اجتناب کنید. با گسترش یافتن شیء در کامپوننت ورودی، شما تمام props ها شامل ویژگی isLoading را انتقال می‌دهید. با این حال، کامپوننت ورودی ممکن است به ویژگی isLoading اهمیت ندهد. شما می‌توانید برای جلوگیری از این اتفاق، از تغییر ساختار بقیه‌ی ES6 استفاده کنید.

src/App.js

```
const withLoading = (Component) => ({ isLoading, ...rest }) =>
isLoading
? <Loading />
: <Component { ...rest } />
```

این کار یک ویژگی را از درون شیء می‌گیرد، اما شیء باقی‌مانده را نگه می‌دارد. این عمل با ویژگی‌های چندگانه به‌خوبی عمل می‌کند. ممکن است پیش از این در مورد destructing assignment[۹۷] چیزهایی خوانده باشید.

اکنون می‌توانید در JSX خود از HOC استفاده کنید. یک مورد استفاده از آن در اپ شما می‌تواند نمایش دگمه‌ی More یا کامپوننت loading باشد. کامپوننت Loading قبلاً در HOC کپسوله شده ، اما کامپوننت ورودی مفقود شده است. در مورد این نمایش کامپوننت Button یا کامپوننت loading، Button همان کامپوننت ورودی برای HOC خواهد بود. کامپوننت خروجی گسترش‌یافته، یک کامپوننت ButtonWithLoading است.

src/App.js

```
const Button = ({
onClick,
className = '',
```

[97] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

```
children,
}) =>
<button
onClick={onClick}
className={className}
type="button"
>
{children}
</button>
const Loading = () =>
<div>Loading ...</div>
const withLoading = (Component) => ({ isLoading, ...rest }) =>
isLoading
? <Loading />
: <Component { ...rest } />
const ButtonWithLoading = withLoading(Button);
```

حالا همه چیز تعریف شده است. به عنوان آخرین مرحله، شما باید از کامپوننت ButtonWithLoading استفاده کنید که state لودینگ را به عنوان یک ویژگی اضافه شده دریافت می‌کند. در حالی که HOC ویژگی لودینگ را مصرف می‌کند، تمام prop های گرفته‌شده‌ی دیگر به Button انتقال داده می‌شوند.

src/App.js

```
class App extends Component {
...
render() {
...
return (
<div className="page">
...
<div className="interactions">
<ButtonWithLoading
isLoading={isLoading}
onClick={() => this.fetchSearchTopStories(searchKey, page + 1)}>
More
</ButtonWithLoading>
</div>
</div>
);
}
}
```

وقتی شما دوباره تست‌های خود را اجرا می‌کنید، متوجه خواهید شد که تست snapshot برای کامپوننت APP شکست خواهد خورد. تفاوت ممکن است شبیه کد زیر در خط فرمان باشد:

Command Line

```
-           <button
-           className=""
-           onClick={[Function]}
-           type="button"
-           >
-           More
-           </button>
+           <div>
+           Loading ...
+           </div>
```

اکنون می‌توانید یا هنگامی که چیزی در مورد کامپوننت اشتباه به نظر می‌رسد، آن را تعمیر کنید، و یا می‌توانید یک snapshot جدید برای آن تأیید کنید. از آن‌جا که کامپوننت Loading را در این گفتار تعریف کرده‌اید، می‌توانید تغییرات تست snapshot در خط فرمان را در تست تعاملی، Export کنید.

کامپوننت‌های مرتبه‌ی بالاتر تکنیک پیشرفته‌ای در ری‌اکت هستند. این کامپوننت‌ها اهداف چندگانه مانند اصلاح قابلیت استفاده‌ی مجدد کامپوننت‌ها، انتزاع بیش‌تر، ترکیب‌پذیری کامپوننت‌ها؛ دستکاری state ها و View را دارند. اگر آن‌ها را به سرعت درک نمی‌کنید، نگران نباشید. برای استفاده از آن‌ها زمان زیادی لازم است.

توصیه می‌کنم مقدمه‌ای کوتاه بر کامپوننت‌های مرتبه‌ی بالاتر[۹۸] را مطالعه کنید. این مقاله به شما یک رویکرد دیگر برای یادگیری کامپوننت‌های مرتبه‌ی بالاتر، و یک روش ظریف برای استفاده از آن‌ها در یک برنامه کاربردی نشان می‌دهد و به طور خاص، مشکل رندر شرطی با کامپوننت‌های مرتبه‌ی بالاتر را حل می‌کند.

تمرین

- مقدمه‌ای کوتاه بر کامپوننت‌های مرتبه‌ی بالاتر[۹۹] را بخوانید.
- با HOC که ایجاد کرده‌اید، آزمایش کنید.
- درباره‌ی موارد استفاده از آن در زمانی که به‌کارگیری یک HOC دیگر هم منطقی باشد فکر کنید.
    - اگر یک مورد استفاده برای HOC وجود دارد، آن را اجرا کنید.

[98] https://www.robinwieruch.de/gentle-introduction-higher-order-components/
[99] https://www.robinwieruch.de/gentle-introduction-higher-order-components/

مرتب‌سازی پیشرفته

شما قبلاً یک تعامل جست‌وجوی سمت سرور و سمت مشتری را اجرا کرده‌اید. از آن‌جایی که شما یک کامپوننت Table دارید، بهتر است آن را با استفاده از تعاملات بهبود دهید. نظرتان در مورد معرفی قابلیت مرتب‌سازی برای هر ستون با استفاده از هِدِر هر ستون Table چیست؟

ممکن است بتوانید تابع مرتب‌سازی خود را بنویسید، اما من شخصاً ترجیح می دهم از یک کتابخانه‌ی سودمند برای چنین مواردی استفاده کنم. Lodash[100] یکی از این کتابخانه‌های مفید است، اما شما می‌توانید از هر کتابخانه‌ی مناسب دیگری استفاده کنید. بیایید ابتدا Lodash را نصب و از آن برای عملکرد مرتب‌سازی استفاده کنیم.

Command Line

```
npm install lodash
```

اکنون می‌توانید Lodash را در فایل src/APP.Js خود ایمپورت کنید.

src/App.js

```
import React, { Component } from 'react';
import axios from 'axios';
import { sortBy } from 'lodash';
import './App.css';
```

چندین ستون در جدول شما وجود دارد. در آن‌ها عنوان، نویسنده، نظرات و امتیازات وجود دارند. شما می‌توانید توابع مرتب‌سازی را تعریف کنید، در حالی که هر تابع یک لیست را می گیرد و لیستی از آیتم های مرتب‌شده بر اساس ویژگی‌های خاص را پس می‌دهد. علاوه بر این، شما به یک تابع مرتب‌سازی پیش‌فرض نیاز دارید که مرتب‌سازی نمی‌کند، بلکه فقط لیست مرتب‌نشده را پس می‌دهد. این state اولیه شما خواهد بود.

src/App.js

```
...
const SORTS = {
NONE: list => list,
TITLE: list => sortBy(list, 'title'),
AUTHOR: list => sortBy(list, 'author'),
COMMENTS: list => sortBy(list, 'num_comments').reverse(),
POINTS: list => sortBy(list, 'points').reverse(),
};
class App extends Component {
...
}
...
```

[100] https://lodash.com

می‌توانید ببینید که دو تابع مرتب‌سازی، یک لیست معکوس را پس می‌دهند. این بدان دلیل است که شما می‌خواهید آیتم‌های دارای بالاترین نظرات و امتیاز را ببینید و نه آیتم‌هایی که وقتی لیست را برای اولین بار مرتب‌سازی می‌کنید کمترین مقدار را داشته باشند.

اکنون شیء SORTS به شما اجازه می‌دهد تمام توابع مرتب‌سازی را ارزیابی کنید.

مجدداً کامپوننت APP شما مسؤول ذخیره‌سازی state در مرتب‌سازی است. state اولیه، تابع مرتب‌سازی پیش‌فرض اولیه خواهد بود که مرتب‌سازی را انجام نمی‌دهد و لیست ورودی را به عنوان خروجی باز می‌گرداند.

src/App.js

```
this.state = {
results: null,
searchKey: '',
searchTerm: DEFAULT_QUERY,
error: null,
isLoading: false,
sortKey: 'NONE',
};
```

هنگامی که شما یک sortKey متفاوت را انتخاب می‌کنید، اجازه دهید آن را AUTHOR key نام‌گذاری کنیم، شما لیست را با تابع مرتب‌سازی مناسب از شیء SORTS مرتب خواهید کرد.

اکنون شما می‌توانید یک متد کلاس جدید را در کامپوننت APP خود تعریف کنید که به سادگی sortKey را در state کامپوننت محلی شما تعریف کند. پس از آن، sortKey را می‌توان برای بازیابی تابع مرتب‌سازی برای اعمال آن در لیست خود استفاده کرد.

src/App.js

```
class App extends Component {
_isMounted = false;
constructor(props) {
...
this.needsToSearchTopStories = this.needsToSearchTopStories.bind(this);
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.fetchSearchTopStories = this.fetchSearchTopStories.bind(this);
this.onSearchSubmit = this.onSearchSubmit.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.onDismiss = this.onDismiss.bind(this);
this.onSort = this.onSort.bind(this);
}
...
onSort(sortKey) {
this.setState({ sortKey });
}
...
}
```

گام بعدی این است که متد و sortKey را به کامپوننت Table انتقال دهید.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey,
error,
isLoading,
sortKey
} = this.state;
...
return (
<div className="page">
...
<Table
list={list}
sortKey={sortKey}
onSort={this.onSort}
onDismiss={this.onDismiss}
/>
...
</div>
);
}
}
```

کامپوننت Table مسؤول مرتب‌سازی لیست است. این کامپوننت یکی از توابع مرتب‌سازی را به وسیله sortKey می‌گیرد و لیست را به عنوان ورودی انتقال می‌دهد. پس از آن، mapping بر روی لیست مرتب‌شده اجرا می‌شود.

src/App.js

```
const Table = ({
list,
sortKey,
onSort,
onDismiss
}) =>
<div className="table">
{SORTS[sortKey](list).map(item =>
<div key={item.objectID} className="table-row">
...
</div>
```

```
)}
</div>
```

به لحاظ تئوری، لیست توسط یکی از توابع مرتب می‌شود. اما مرتب‌سازی پیش‌فرض بر روی NONE تنظیم شده است، بنابراین هنوز هیچ مرتب‌سازی‌ای صورت نگرفته است. تاکنون هیچ کس متد ()onSort را برای تغییر sortKey اجرا نکرده است. اجازه دهید Table را با یک ردیف از هدرهای ستون گسترش دهیم تا از کامپوننت Sort برای مرتب‌سازی هر ستون استفاده کند.

src/App.js

```
const Table = ({
list,
sortKey,
onSort,
onDismiss
}) =>
<div className="table">
<div className="table-header">
<span style={{ width: '40%' }}>
<Sort
sortKey={'TITLE'}
onSort={onSort}
>
Title
</Sort>
</span>
<span style={{ width: '30%' }}>
<Sort
sortKey={'AUTHOR'}
onSort={onSort}
>
Author
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'COMMENTS'}
onSort={onSort}
>
Comments
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'POINTS'}
onSort={onSort}
>
Points
</Sort>
```

```
</span>
<span style={{ width: '10%' }}>
Archive
</span>
</div>
{SORTS[sortKey](list).map(item =>
...
)}
</div>
```

هر کامپوننت Sort یک sortKey خاص و تابع کلی ()onSort را دریافت می‌کند. در داخل آن برای یک کلید خاص یک متد با sortKey فراخوانی می‌شود.

src/App.js

```
const Sort = ({ sortKey, onSort, children }) =>
<Button onClick={() => onSort(sortKey)}>
{children}
</Button>
```

همان‌طور که می بینید، کامپوننت Sort مجدداً از کامپوننت Button شما استفاده می‌کند. اکنون وقتی روی هدرهای ستون کلیک می‌کنید، باید بتوانید لیست را مرتب کنید.

در این‌جا یک بهبود جزئی برای ظاهر بهتر وجود دارد. تاکنون، دگمه‌ی درون ستون هدر کمی مسخره به نظر می‌رسد. بیایید به دگمه‌ی موجود در کامپوننت sort ، ویژگی Class name بدهیم.

src/App.js

```
const Sort = ({ sortKey, onSort, children }) =>
<Button
onClick={() => onSort(sortKey)}
className="button-inline"
>
{children}
</Button>
```

حالا باید ظاهر خوبی پیدا کرده باشد. هدف بعدی پیاده‌سازی مرتب‌سازی معکوس است. هنگامی که برای دومین بار روی کامپوننت Sort کلیک می‌کنید، لیست باید به صورت معکوس مرتب‌سازی شود. ابتدا لازم است یک state معکوس را با یک boolean تعریف کنید. مرتب‌سازی می‌تواند به صورت معکوس یا غیر معکوس باشد.

src/App.js

```
this.state = {
results: null,
```

```
searchKey: '',
searchTerm: DEFAULT_QUERY,
error: null,
isLoading: false,
sortKey: 'NONE',
isSortReverse: false,
};
```

اکنون شما می‌توانید در متد Sort خود، امکان مرتب‌سازی معکوس یک لیست را ارزیابی کنید. این لیست در صورتی معکوس است که sortKey در داخل state، مشابه sortKey ورودی باشد و state معکوس پیش از این روی true تنظیم نشده باشد.

src/App.js

```
onSort(sortKey) {
const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortReve\
rse;
this.setState({ sortKey, isSortReverse });
}
```

مجدداً می‌توانید prop معکوس را به کامپوننت Table خود انتقال دهید.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey,
error,
isLoading,
sortKey,
isSortReverse
} = this.state;
...
return (
<div className="page">
...
<Table
list={list}
sortKey={sortKey}
isSortReverse={isSortReverse}
onSort={this.onSort}
onDismiss={this.onDismiss}
/>
```

```
...
</div>
);
}
}
```

اکنون Table باید برای محاسبه‌ی داده‌ها یک تابع arrow داشته باشد.

src/App.js

```
const Table = ({
list,
sortKey,
isSortReverse,
onSort,
onDismiss
}) => {
const sortedList = SORTS[sortKey](list);
const reverseSortedList = isSortReverse
? sortedList.reverse()
: sortedList;
return(
<div className="table">
<div className="table-header">
...
</div>
{reverseSortedList.map(item =>
...
)}
</div>
);
}
```

مرتب‌سازی معکوس اکنون باید کار کند.

در نهایت، شما باید برای بهبود تجربه‌ی کاربری ، با یک سؤال باز درگیر شوید. آیا کاربر می‌تواند تشخیص دهد مرتب‌سازی کدام ستون فعال است؟ این امکان تا این‌جا وجود نداشت. بیایید به کاربر یک بازخورد بصری بدهیم.

اکنون هر کامپوننت Sort یک sortKey خاص را دریافت می‌کند. این مسأله می‌تواند برای شناسایی مرتب‌سازی فعال مورد استفاده قرار گیرد. شما می‌توانید sortKey را به عنوان کلید مرتب‌سازی فعال، از state کامپوننت داخلی به کامپوننت Sort انتقال دهید.

src/App.js

```
const Table = ({
list,
sortKey,
```

```
isSortReverse,
onSort,
onDismiss
}) => {
const sortedList = SORTS[sortKey](list);
const reverseSortedList = isSortReverse
? sortedList.reverse()
: sortedList;
return(
<div className="table">
<div className="table-header">
<span style={{ width: '40%' }}>
<Sort
sortKey={'TITLE'}
onSort={onSort}
activeSortKey={sortKey}
>
Title
</Sort>
</span>
<span style={{ width: '30%' }}>
<Sort
sortKey={'AUTHOR'}
onSort={onSort}
activeSortKey={sortKey}
>
Author
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'COMMENTS'}
onSort={onSort}
activeSortKey={sortKey}
>
Comments
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'POINTS'}
onSort={onSort}
activeSortKey={sortKey}
>
Points
</Sort>
</span>
<span style={{ width: '10%' }}>
Archive
</span>
</div>
{reverseSortedList.map(item =>
```

```
...
)}
</div>
);
}
```

اکنون در کامپوننت Sort بر اساس sortKey و active sortKey می‌توانید تشخیص دهید که آیا مرتب‌سازی فعال است یا خیر. برای ارائه‌ی یک بازخورد بصری به کاربر، هنگامی که مرتب‌سازی فعال است، یک ویژگی className اضافی به کامپوننت Sort بدهید.

src/App.js

```
const Sort = ({
sortKey,
activeSortKey,
onSort,
children
}) => {
const sortClass = ['button-inline'];
if (sortKey === activeSortKey) {
sortClass.push('button-active');
}
return (
<Button
onClick={() => onSort(sortKey)}
className={sortClass.join(' ')}
>
{children}
</Button>
);
}
```

روش تعریف sortClass کمی شلخته به نظر می‌رسد، این طور نیست؟ یک کتابخانه‌ی کوچک شسته‌رفته وجود دارد که از این حالت خلاص شوید. ابتدا باید آن را نصب کنید.

Command Line

```
npm install classnames
```

و در مرحله‌ی دوم باید آن را در فایل src/APP.js ایمپورت کنید.

src/App.js

```
import React, { Component } from 'react';
import axios from 'axios';
```

```
import { sortBy } from 'lodash';
import classNames from 'classnames';
import './App.css';
```

اکنون می‌توانید از آن برای تعریف className کامپوننت خود با کلاس‌های شرطی استفاده کنید.

src/App.js

```
const Sort = ({
sortKey,
activeSortKey,
onSort,
children
}) => {
const sortClass = classNames(
'button-inline',
{ 'button-active': sortKey === activeSortKey }
);
return (
<Button
onClick={() => onSort(sortKey)}
className={sortClass}
>
{children}
</Button>
);
}
```

مجدداً، وقتی شما تست‌هایتان را اجرا می‌کنید، باید تست‌های snapshot شکست‌خورده‌ی خود را ببینید. همچنین تست‌های واحد برای کامپوننت Table هم fail خواهند بود. از آن‌جایی که مجدداً نمایش کامپوننت خود را تغییر داده‌اید، می‌توانید تست‌های snapshot را تأیید کنید. اما تست واحد را باید تعمیر نمایید. در فایل src/APP.js نیاز دارید یک sortKey و همچنین یک بولین isSortReverse برای کامپوننت Table ایجاد کنید.

src/App.test.js

```
...
describe('Table', () => {
const props = {
list: [
{ title: '1', author: '1', num_comments: 1, points: 2, objectID: 'y' },
{ title: '2', author: '2', num_comments: 1, points: 2, objectID: 'z' },
],
sortKey: 'TITLE',
isSortReverse: false,
};
...
});
```

ممکن است مجدداً نیاز داشته باشید تست‌های sortKey که fail شده اند را برای کامپوننت Table تأیید کنید، زیرا props های گسترش‌یافته را برای کامپوننت Table ارائه کرده‌اید.

در نهایت، تعامل مرتب‌سازی پیشرفته‌ی شما اکنون کامل است.

تمرین:

- از یک کتابخانه مانند Font Awesome[۱۰۱] برای نمایش مرتب‌سازی (معکوس) استفاده کنید.
  - این کتابخانه می‌تواند یک فلش روبه‌بالا یا روبه‌پایین بعد از هر هدر Sort باشد.
- در مورد کتابخانه‌ی classnames[۱۰۲] بیش‌تر بخوانید.

[101] http://fontawesome.io
[102] https://github.com/JedWatson/classnames

شما تکنیک‌های کامپوننت پیشرفته را در ری‌اکت یاد گرفتید! بیایید گفتار را مرور کنیم:

- **ری‌اکت :**
  - ویژگی ref برای ارجاع به نودهای DOM
  - کامپوننت‌های مرتبه‌ی بالاتر یک راه معمول برای ساخت کامپوننت‌های پیشرفته هستند.
  - اجرای تعاملات پیشرفته در ری‌اکت.
  - className های شرطی با کمک یک کتابخانه‌ی شسته‌رفته.

- **ES6:**
  - تغییر ساختار بقیه برای تقسیم اشیاء و آرایه‌ها

شما می‌توانید سورس کد را در ریپازیتوری رسمی[۱۰۳] پیدا کنید.

[103] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.5

مدیریت state در ری‌اکت و فراتر از آن

در گفتارهای قبل، اصول مدیریت state در ری‌اکت را فرا گرفتید. این گفتار کمی عمیق‌تر به این موضوع می‌پردازد. شما بهترین روش‌ها، چگونگی استفاده از آن‌ها و دلیل این که می‌توانید از یک کتابخانه‌ی مدیریت state شخص ثالث استفاده کنید را یاد خواهید گرفت.

حالت lifting

فقط کامپوننت APP در اپ شما یک کامپوننت stateful ES6 است. این کامپوننت تعداد زیادی state در اپ و logic در متدهای کلاس را مدیریت می‌کند. شاید متوجه شده باشید که شما تعداد زیادی ویژگی را به کامپوننت Table انتقال می‌دهید. بسیاری از این props ها فقط در کامپوننت Table استفاده می‌شوند. در نتیجه می‌توان استدلال کرد که منطقی نیست کامپوننت APP در مورد آن‌ها بداند.

تمام قابلیت مرتب‌سازی فقط در کامپوننت Table استفاده می‌شود. شما می‌توانید آن را به کامپوننت Table منتقل کنید، زیرا کامپوننت APP نیازی به شناختن همه‌ی آن‌ها ندارد. فرآیند اصلاح کردن زیرحالت از یک کامپوننت به کامپوننت دیگر به عنوان *حالت lifting* شناخته می‌شود. در مورد اپ شما، می‌خواهید state هایی که در کامپوننت Table استفاده نمی‌شوند را به کامپوننت Table منتقل کنید. state از والد به فرزند انتقال پیدا می‌کند.

برای تعامل با state و متدهای کلاس در کامپوننت Table باید آن را به کامپوننت کلاس ES6 تبدیل کنید. بازسازی از کامپوننت بدون state کاربردی به کامپوننت کلاس ES6 ساده است.

کامپوننت Table شما به عنوان یک کامپوننت بدون state کاربردی:

src/App.js

```
const Table = ({
list,
sortKey,
isSortReverse,
onSort,
onDismiss
}) => {
const sortedList = SORTS[sortKey](list);
const reverseSortedList = isSortReverse
? sortedList.reverse()
: sortedList;
return(
...
);
}
```

کامپوننت Table شما به عنوان یک کامپوننت کلاس ES6:

src/App.js

```
class Table extends Component {
render() {
const {
list,
sortKey,
isSortReverse,
onSort,
onDismiss
```

```
} = this.props;
const sortedList = SORTS[sortKey](list);
const reverseSortedList = isSortReverse
? sortedList.reverse()
: sortedList;
return (
...
);
}
}
```

از آن‌جا که شما می‌خواهید با state و متدها در کامپوننت تعامل داشته باشید، باید سازنده و state اولیه را اضافه کنید.

src/App.js

```
class Table extends Component {
constructor(props) {
super(props);
this.state = {};
}
render() {
...
}
}
```

حالا می‌توانید state و متدهای کلاس که به عملکرد مرتب‌سازی مربوط هستند را از کامپوننت APP به کامپوننت Table منتقل کنید.

src/App.js

```
class Table extends Component {
constructor(props) {
super(props);
this.state = {
sortKey: 'NONE',
isSortReverse: false,
};
this.onSort = this.onSort.bind(this);
}
onSort(sortKey) {
const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortRe\
verse;
this.setState({ sortKey, isSortReverse });
}
render() {
...
}
}
```

فراموش نکنید که state و متد کلاس ()onSort که منتقل کرده‌اید را از کامپوننت APP حذف کنید.

src/App.js

```
class App extends Component {
_isMounted = false;
constructor(props) {
super(props);
this.state = {
results: null,
searchKey: '',
searchTerm: DEFAULT_QUERY,
error: null,
isLoading: false,
};
this.setSearchTopStories = this.setSearchTopStories.bind(this);
this.fetchSearchTopStories = this.fetchSearchTopStories.bind(this);
this.onDismiss = this.onDismiss.bind(this);
this.onSearchSubmit = this.onSearchSubmit.bind(this);
this.onSearchChange = this.onSearchChange.bind(this);
this.needsToSearchTopStories = this.needsToSearchTopStories.bind(this);
}
...
}
```

علاوه بر این، می‌توانید API کامپوننت Table را سبک‌تر کنید. props هایی که از کامپوننت APP به آن انتقال داده شده‌اند را حذف کنید، زیرا اکنون آن‌ها در داخل کامپوننت Table کار می‌کنند.

src/App.js

```
class App extends Component {
...
render() {
const {
searchTerm,
results,
searchKey,
error,
isLoading
} = this.state;
...
return (
<div className="page">
...
{ error
? <div className="interactions">
<p>Something went wrong.</p>
</div>
```

```
: <Table
list={list}
onDismiss={this.onDismiss}
/>
}
...
</div>
);
}
}
```

اکنون در کامپوننت Table می‌توانید از متد ()onSort و state داخلی Table استفاده کنید.

src/App.js

```
class Table extends Component {
...
render() {
const {
list,
onDismiss
} = this.props;
const {
sortKey,
isSortReverse,
} = this.state;
const sortedList = SORTS[sortKey](list);
const reverseSortedList = isSortReverse
? sortedList.reverse()
: sortedList;
return(
<div className="table">
<div className="table-header">
<span style={{ width: '40%' }}>
<Sort
sortKey={'TITLE'}
onSort={this.onSort}
activeSortKey={sortKey}
>
Title
State Management in React and beyond 180
</Sort>
</span>
<span style={{ width: '30%' }}>
<Sort
sortKey={'AUTHOR'}
onSort={this.onSort}
activeSortKey={sortKey}
>
Author
```

```
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'COMMENTS'}
onSort={this.onSort}
activeSortKey={sortKey}
>
Comments
</Sort>
</span>
<span style={{ width: '10%' }}>
<Sort
sortKey={'POINTS'}
onSort={this.onSort}
activeSortKey={sortKey}
>
Points
</Sort>
</span>
<span style={{ width: '10%' }}>
Archive
</span>
</div>
{ reverseSortedList.map((item) =>
...
)}
</div>
);
}
}
```

اپ شما هنوز هم باید کار کند. اما شما یک اصلاح بسیار مهم انجام داده‌اید. شما عملکرد و state را به یک کامپوننت دیگر انتقال داده‌اید. کامپونت‌های دیگر دوباره سبک‌تر شدند. علاوه بر این، API کامپونت Table بیش‌تر سبک شده است، زیرا با ویژگی‌های مرتب‌سازی به صورت داخلی در ارتباط است.

فرآیند حالت lifting می‌تواند به شیوه‌ی دیگری نیز انجام شود: از فرزند به کامپوننت والد. این فرایند حالت lifting up نامیده می‌شود. تصور کنید با یک state داخلی در یک کامپوننت فرزند برخورد کردید. حالا می‌خواهید یک درخواست برای نمایش state در داخل کامپوننت والد انجام دهید. شما باید state را به کامپوننت والد lift up کنید. این حتی فراتر هم می‌رود. تصور کنید که می‌خواهید state را داخل یک کامپوننت خواهر یک کامپوننت فرزند نمایش دهید. مجدداً باید state را به کامپوننت والد lift up کنید. کامپوننت والد با state در ارتباط است، اما آن را معرض هر دو کامپوننت فرزند قرار می‌دهد.

تمرین

- در مورد lifting state در ری‌اکت[۱۰۴] بیش‌تر بخوانید.
- در مورد lifting state در یادگیری ری‌اکت قبل از استفاده ریداکس[۱۰۵] بیش‌تر بخوانید.

[104] https://reactjs.org/docs/lifting-state-up.html
[105] https://www.robinwieruch.de/learn-react-before-using-redux/

مرور دوباره: setState()

تا این‌جا، شما از setState() ری‌اکت برای مدیریت state کامپوننت داخلی استفاده کرده‌اید. می‌توانید یک شیء را به تابعی انتقال دهید که در آن می‌توانید state داخلی را به‌روز کنید.

Code Playground

```
this.setState({ foo: bar });
```

اما setState() فقط یک شیء را نمی‌گیرد. در نسخه‌ی دوم، شما می‌توانید یک تابع را برای به‌روزرسانی state انتقال دهید.

Code Playground

```
this.setState((prevState, props) => {
...
});
```

چرا ممکن است بخواهید این کار را انجام دهید؟ یک مورد استفاده مهم وجود دارد که در آن کاربرد یک تابع به جای شیء منطقی است. این زمانی‌ست که یک state را با توجه به state یا props قبلی به‌روزرسانی می‌کنید. در این حالت، اگر از یک تابع استفاده نکنید، مدیریت state داخلی می‌تواند باعث ایجاد خطا شود.

اما چرا زمانی که به‌روزرسانی وابسته به state یا props قبلی است، استفاده از یک شیء به جای تابع باعث ایجاد خطا می‌شود؟ متد setState() در ری‌اکت ناهمگام است. ری‌اکت دسته‌ای از setState() را فراخوانی و در نهایت آن‌ها را اجرا می‌کند. ممکن است زمانی که بر فراخوانی setState() خود تکیه می‌کنید، این اتفاق بیافتد که state یا props قبلی تغییر کند.

Code Playground

```
const { fooCount } = this.state;
const { barCount } = this.props;
this.setState({ count: fooCount + barCount });
```

تصور کنید که fooCount و barCount، و در نتیجه state یا props، وقتی شما setState() را فراخوانی می‌کنید، جای دیگری به صورت ناهمگام تغییر کند. در یک اپ در حال رشد، شما بیش از یک فراخوانی setState() در اپ خود دارید. از آنجا که setState() به صورت ناهمگام اجرا می‌شود، در این مثال می‌توانید به مقدار state تکیه کنید.

با استفاده از رویکرد تابع، تابع در setState() یک فراخوان مجدد است که در زمان اجرای تابع فراخوان مجدد بر روی state و props اجرا می‌شود. اگرچه setState() ناهمگام است، با یک تابع در زمان اجرا می‌تواند state و props را بگیرد.

Code Playground

```
this.setState((prevState, props) => {
const { fooCount } = prevState;
const { barCount } = props;
return { count: fooCount + barCount };
});
```

اکنون اجازه دهید به کد برگردیم تا این رفتار را اصلاح کنیم. جایی که set state استفاده می‌شود و به state یا props تکیه می‌کند، کدها را با هم اصلاح می کنیم. بعداً می‌توانید آن را در مکان های دیگر نیز اصلاح کنید.

متد setSearchTopStories() به state قبلی متکی است و در نتیجه یک مثال کامل برای استفاده از تابع در اولویت نسبت به شیء می‌باشد. در حال حاضر، کد به‌طور خلاصه به شکل زیر به‌نظر می رسد:

src/App.js

```
setSearchTopStories(result) {
const { hits, page } = result;
const { searchKey, results } = this.state;
const oldHits = results && results[searchKey]
? results[searchKey].hits
: [];
const updatedHits = [
...oldHits,
...hits
];
this.setState({
results: {
...results,
[searchKey]: { hits: updatedHits, page }
},
isLoading: false
});
}
```

شما مقادیر را از state استخراج می‌کنید، اما به‌روزرسانی state به صورت ناهمگام وابسته به مقدار state قبلی است. حالا می‌توانید از رویکرد تابع برای جلوگیری از خطا استفاده کنید، زیرا state دائمی است.

src/App.js

```
setSearchTopStories(result) {
const { hits, page } = result;
this.setState(prevState => {
...
});
}
```

می‌توانید همه‌ی بلوک را که قبلاً در تابع اجرا کرده‌اید، جابه‌جا کنید. شما فقط باید this.state را به prevState تغییر دهید.

src/App.js

```
setSearchTopStories(result) {
const { hits, page } = result;
this.setState(prevState => {
const { searchKey, results } = prevState;
const oldHits = results && results[searchKey]
? results[searchKey].hits
: [];
const updatedHits = [
...oldHits,
...hits
];
return {
results: {
...results,
[searchKey]: { hits: updatedHits, page }
},
isLoading: false
};
});
}
```

این کار مسأله‌ی state دائمی را حل می‌کند. در این‌جا یک بهبود دیگر نیز وجود دارد. از آن‌جا که این یک تابع است، شما می‌توانید تابع را برای خوانایی بیش‌تر extract کنید. این یک مزیت دیگر استفاده از تابع ارجح بر شیء است. این تابع می‌تواند در خارج از کامپوننت هم وجود داشته باشد. اما شما باید از یک تابع مرتبه‌ی بالاتر استفاده کنید تا نتیجه را به آن انتقال دهید. در نهایت، شما می‌خواهید state را بر اساس نتیجه‌ی دریافت‌شده از API به‌روزرسانی کنید.

src/App.js

```
setSearchTopStories(result) {
const { hits, page } = result;
this.setState(updateSearchTopStoriesState(hits, page));
}
```

تابع updateSearchTopStoriesState() باید یک تابع را بازگردانی کند. این یک تابع مرتبه‌ی بالاتر است. شما می‌توانید این تابع مرتبه‌ی بالاتر را خارج از کامپوننت APP خود ایجاد کنید. توجه داشته باشید که در این‌جا این تابع کمی تغییر می‌کند.

src/App.js

```
const updateSearchTopStoriesState = (hits, page) => (prevState) => {
const { searchKey, results } = prevState;
```

```
const oldHits = results && results[searchKey]
? results[searchKey].hits
: [];
const updatedHits = [
...oldHits,
...hits
];
return {
results: {
...results,
[searchKey]: { hits: updatedHits, page }
},
isLoading: false
};
};
class App extends Component {
...
}
```

تمام شد. استفاده از تابع به جای شیء در ()setState باعث رفع خطاهای بالقوه می‌شود و در عین حال خوانایی و قابلیت نگهداری شما را افزایش می‌دهد. علاوه بر این، این تابع خارج از کامپوننت APP نیز قابل استفاده می‌شود. شما می‌توانید آن را Export کنید و به عنوان تمرین از آن استفاده کنید.

تمرین:

- در مورد استفاده‌ی درست state در ری‌اکت[106] بیش‌تر بخوانید.
- تابع updateSearchTopStoriesState را از فایل Export کنید.
- یک تست برای آن بنویسید و payload (بازدیدها و صفحه) را به آن انتقال دهید و یک state قبلی ایجاد کنید و در نهایت، منتظر یک state جدید باشید.
- متدهای ()setState برای استفاده از تابع را بازنویسی کنید.
    - اما فقط در جایی که ساختن آن منطقی است، چون این متد بر state یا props متکی است.
- تست‌های خود را مجدداً اجرا کنید و مطمئن شوید همه چیز روبه‌راه است.

[106] https://reactjs.org/docs/state-and-lifecycle.html#using-state-correctly

مهار کردن state

گفتار های قبل به شما نشان دادند که مدیریت state می‌تواند یک موضوع مهم در اپ‌های بزرگ‌تر باشد. به طور کلی نه تنها ری‌اکت، بلکه بسیاری از فریم‌ورک های SPA با آن دست‌وپنجه نرم می‌کنند. اپ‌ها در سال‌های اخیر پیچیده‌تر شده‌اند. امروزه یک چالش بزرگ در اپ‌های وب، کنترل state هاست.

در مقایسه با راه حل‌های دیگر، ری‌اکت در حال حاضر یک قدم بزرگ به جلو پیش رفته است. جریان داده‌ی یک‌طرفه و API ساده برای مدیریت state در یک کامپوننت ضروری هستند. این مفاهیم باعث آسان‌تر شدن نتایج در مورد state و تغییر state شما می‌شود. این باعث می‌شود استدلال در مورد state و تغییر state های شما آسان‌تر شود. این امر موجب می‌شود استدلال در مورد سطح کامپوننت و به صورت عمیق‌تر، سطح اپ، آسان‌تر باشد.

در یک اپ در حال رشد، درک تغییر state سخت‌تر می‌شود. زمانی که از یک شیء ارجح بر تابع در ()setState استفاده می‌کنید، می‌توانید باگ‌ها را با انجام عملیات در state قدیمی معرفی کنید. شما باید برای به اشتراک گذاشتن state مورد نیاز یا مخفی کردن state غیرضروری در سراسر کامپوننت state را بالا ببرید.

این اتفاق می‌تواند زمانی بیافتد که در کامپوننت نیاز به lift up کردن state وجود دارد، چون کامپوننت خواهر او، به آن وابسته است. شاید کامپوننت در درخت کامپوننت (Component tree) دور باشد و بنابراین شما باید state را در سراسر درخت کامپوننت به اشتراک بگذارید. در نتیجه کامپوننت‌ها با گسترش یافتن در مدیریت state ها دچار پیچیدگی می‌شوند. اما در نهایت، مسؤولیت اصلی کامپوننت باید نمایش UI باشد. آیا این چنین نیست؟

به‌خاطر تمام این دلایل، راه حل‌های مستقل برای مدیریت state ها وجود دارد. این راه حل‌ها تنها در ری‌اکت استفاده نمی‌شوند. با این حال، این همان چیزی است که در اکوسیستم ری‌اکت این امکان قدرتمند را ایجاد می‌کند. شما می‌توانید از راه حل‌های مختلف برای حل مشکلات خود استفاده کنید. برای حل مشکل مدیریت state ها، ممکن است در مورد کتابخانه Redux[۱۰۷] یا MobX[۱۰۸] شنیده باشید. شما می‌توانید از هر یک از این راه حل‌ها در یک اپ ری‌اکت استفاده کنید. آن‌ها با افزونه‌ها، react-redux و MobX-react می‌آیند تا آن‌ها را در لایه view جدید ری‌اکت ادغام کنند.

Redux و Mob X خارج از محدوده‌ی این کتاب هستند. هنگامی که این کتاب را به پایان رساندید، به شما راهنمایی می‌کنیم که چگونه می‌توانید به یادگیری ری‌اکت و اکوسیستم آن ادامه دهید. یک مسیر یادگیری می‌تواند یادگیری Redux باشد. قبل از این که به موضوع مدیریت state خارجی عمیقاً وارد شوید، می‌توانم این مقاله[۱۰۹] را برای خواندن توصیه کنم. این به شما درک بهتری برای چگونگی یادگیری مدیریت state خارجی می‌دهد.

تمرین

- در مورد مدیریت state خارجی و نحوه یادگیری آن[۱۱۰] بیش‌تر بخوانید.
- کتاب دوم من در مورد مدیریت state در ری‌اکت[۱۱۱] را مطالعه کنید.

---

[107] https://redux.js.org/introduction/getting-started
[108] https://mobx.js.org
[109] https://www.robinwieruch.de/redux-mobx-confusion/
[110] https://www.robinwieruch.de/redux-mobx-confusion/

شما صورت پیشرفته state در ری‌اکت را یاد گرفته‌اید! بیایید گفتار گذشته را مرور کنیم:

- **ری‌اکت**
  - Lift کردن state به بالا و پایین برای مناسب شدن کامپوننت
  - setState می‌تواند از یک تابع برای جلوگیری از باگ‌های موجود استفاده کند.
  - راه حل‌های خارجی موجود به شما کمک می‌کند تا state را مهار کنید.

شما می‌توانید سورس کدها را در ریپازیتوری رسمی[۱۱۲] من پیدا کنید.

[111] https://roadtoreact.com
[112] https://github.com/the-road-to-learn-react/hackernews-client/tree/5.6

مراحل نهایی تولید

آخرین گفتارها به شما نشان می‌دهند که چگونه اپ خود را برای تولید، گسترش دهید. شما از سرویس میزبانی رایگان Heroku استفاده خواهید کرد. در مورد راه‌اندازی اپ، باید بیش‌تر در مورد create-react-app بیاموزید.

Eject

مرحله‌ی زیر و دانستن آن، برای گسترش اپ شما برای تولید، **ضروری نیست**. با این حال من می‌خواهم آن را به شما توضیح دهم. Create-react-app با یک ویژگی برای توسعه‌پذیر بودن همراه است، اما برای جلوگیری از قفل شدن نرم‌افزار نیز استفاده می‌شود. قفل شدن نرم‌افزار معمولاً هنگامی اتفاق می‌افتد که شما یک تکنولوژی را خریداری می‌کنید، اما هیچ راه میان‌بری برای استفاده از آن در آینده وجود ندارد. خوشبختانه در create-react-app شما این راه گریز را با استفاده از eject دارید.

در فایل package.json شما اسکریپت‌هایی برای شروع، تست و ساخت اپ خود پیدا خواهید کرد. آخرین اسکریپت eject است. می‌توانید آن را امتحان کنید، اما هیچ راه برگشتی وجود ندارد. **این عملیات یک‌طرفه است. پس از eject شما نمی‌توانید به عقب برگردید!** اگر شما به‌تازگی یادگیری ری‌اکت را شروع کرده اید، دلیلی منطقی برای رها کردن محیط راحت create-react-app وجود ندارد.

اگر شما دستور npm run eject را اجرا کنید، این دستور تمام پیکربندی‌ها و نیازمندی‌ها را در package.json و یک فولدر جدید config کپی می‌کنید. شما می‌توانید کل پروژه را به یک ساختار سفارشی با ابزارهایی شامل Babel و webpack تبدیل کنید. به هر حال، می‌توانید کنترل کاملی بر تمام این ابزارها داشته باشید.

اسناد رسمی می گویند که create-react-app برای ایجاد اپ‌های کوچک تا متوسط مناسب هستند. شما نباید برای استفاده از دستور eject احساس تکلیف کنید.

تمرین

- در مورد eject[113] بیش‌تر بخوانید.

[113] https://github.com/facebook/create-react-app#converting-to-a-custom-setup

اپ خود را مستقر کنید

در نهایت، هیچ اپی نباید بر روی Localhost باقی بماند. شما می‌خواهید به آن زندگی ببخشید. Heroku یک پلتفرم خدماتی است که می‌تواند اپ شما را میزبانی کند. این پلتفرم با ری‌اکت تجمیع کاملی دارد. اگر بخواهیم دقیق‌تر بگوییم: تنها در چند دقیقه می‌توان create-react-app را راه‌اندازی کرد. این یک راه‌اندازی صفر-پیکربندی است که فلسفه create-react-app را دنبال می‌کند.

قبل از این که بتوانید اپ خود را به Heroku بفرستید، باید دو پیش‌نیاز را در نظر بگیرید:

- نصب Heroku CLI[۱۱۴]
- ساخت یک اکانت رایگان Heroku[۱۱۵]

اگر شما Hombrew را نصب کرده‌اید، می‌توانید Heroku CLI را از خط فرمان نصب کنید:

Command Line

```
brew update
brew install heroku-toolbelt
```

اکنون می‌توانید از git و Heroku CLI برای گسترش اپ خود استفاده کنید.

Command Line

```
git init
heroku create -b https://github.com/mars/create-react-app-buildpack.git
git add .
git commit -m "react-create-app on Heroku"
git push heroku master
heroku open
```

تمام شد. امیدوارم اپ شما اکنون در حال اجرا باشد. اگر با مشکلی مواجه شدید، می‌توانید منابع زیر را بررسی کنید:

- Git and Git Hub Essential[۱۱۶]
- استقرار ری‌اکت با صفر- پیکربندی[۱۱۷]
- Heroku Build pack برای create-react-app[۱۱۸]

[114] https://devcenter.heroku.com/articles/heroku-cli
[115] https://www.heroku.com
[116] https://www.robinwieruch.de/git-essential-commands/
[117] https://blog.heroku.com/deploying-react-with-zero-configuration
[118] https://github.com/mars/create-react-app-buildpack

خلاصه

این آخرین گفتار کتاب بود. امیدوارم از آن لذت برده باشید و به شما کمک کرده باشد تا ری‌اکت را کشف کنید. اگر کتاب را دوست داشتید، آن را به عنوان راهی برای یادگیری ری‌اکت با دوستانتان به اشتراک بگذارید. این کتاب باید خواند و زنجیره‌وار به دیگران انتقال داد. علاوه بر این، اگر به مدت ۵ دقیقه زمان بگذارید و در مورد این کتاب در آمازون[۱۱۹] یا Goodreads[۱۲۰] بنویسید، برای من بسیار ارزشمند خواهد بود.

قبل از اینکه به سراغ کتاب دیگری بروید، چه کتاب درسی یا خودآموز، باید پروژه‌ی ری‌اکت خود را به‌صورت دستی ایجاد کنید. این پروژه را در یک هفته انجام دهید، آن را جایی راه‌اندازی کنید، و سپس برای من[۱۲۱] یا دیگران بفرستید تا آن را بررسی کنیم. من کنجکاو هستم ببینم پس از این که این کتاب را خواندید، چه چیزی خواهید ساخت.

اگر به دنبال ویژگی‌های بیش‌تری برای اپ خود هستید، می‌توانم چند مسیر یادگیری را پس از خواندن این کتاب به شما ارائه دهم:

- **مدیریت state:** شما از this.setState() و this.state برای مدیریت و دسترسی به state کامپوننت داخلی استفاده کرده‌اید. این یک شروع عالی است. با این حال، در یک اپ بزرگتر، محدودیت‌های state داخلی کامپوننت در ری‌اکت[۱۲۲] را تجربه خواهید کرد. بنابراین می‌توانید از یک کتابخانه‌ی مدیریت state شخص سوم مثل redux یا MobX[۱۲۳] استفاده کنید. در پلتفرم دوره‌ی مسیر ری‌اکت[۱۲۴] شما «مهار state در ری‌اکت» را پیدا خواهید کرد که به صورت پیشرفته state محلی در ری‌اکت، redux و MobX را تدریس می‌کند.
- **اتصال به پایگاه داده / یا اعتبار سنجی**: در یک اپ ری‌اکت در حال رشد، در نهایت بخواهید داده‌ها را حفظ کنید. داده‌ها باید در یک پایگاه داده ذخیره شوند تا پس از جلسه‌ی مرورگر بتوانند باقی بمانند و در میان کاربرانی که از اپ شما استفاده می‌کنند، به اشتراک گذاشته شوند. ساده‌ترین راه برای معرفی یک پایگاه داده، استفاده از Firebase است. در «این آموزش جامع[۱۲۵]» یک راهنمای گام‌به‌گام در مورد نحوه‌ی استفاده از اعتبارسنجی Firebase (ثبت نام، ورود به سیستم، خروج و...) در ری‌اکت را در اختیار خواهید داشت. فراتر از آن، شما از پایگاه داده‌ی Firebase برای ذخیره‌ی اطلاعات کاربر استفاده خواهید کرد. پس از آن، ذخیره داده‌های بیش‌تر در پایگاه داده که مورد نیاز اپ شماست، یکی از انتخاب‌های شما خواهد بود.
- **تجهیز با web pack و Babel:** در این کتاب برای تنظیمات اپ خود از creat-react-app استفاده کردید. در بعضی موارد، هنگامی که ری‌اکت را یاد گرفتید، ممکن است بخواهید ابزارهای جانبی آن را بیاموزید. این شما را قادر می‌سازد اپ خود را بدون نیاز به creat-react-app ایجاد کنید. به شما توصیه می کنم حداقل تنظیمات مربوط به webpack و Babel[۱۲۶] را یاد بگیرید. پس از آن می‌توانید ابزارهای بیش‌تری را به تنهایی اعمال کنید. به عنوان مثال، می‌توانید از ESlint[۱۲۷] برای دنبال کردن یک سبک کد یکپارچه در اپ خود استفاده کنید.

---

119 https://www.amazon.com/dp/B077HJFCQX
120 https://www.goodreads.com/book/show/37503118-the-road-to-learn-react
121 https://twitter.com/rwieruch
122 https://www.robinwieruch.de/learn-react-before-using-redux/
123 https://www.robinwieruch.de/redux-mobx-confusion/
124 https://roadtoreact.com
125 https://www.robinwieruch.de/complete-firebase-authentication-react-tutorial/
126 https://www.robinwieruch.de/minimal-react-webpack-babel-setup/
127 https://www.robinwieruch.de/react-eslint-webpack-babel/

- **Syntax کامپوننت ری‌اکت:** احتمالات و بهترین روش‌ها برای پیاده‌سازی کامپوننت‌های ری‌اکت در طول زمان تکامل یافته است. شما می‌توانید راه‌های بسیاری برای نوشتن کامپوننت‌های ری‌اکت خود، به‌خصوص کلاس کامپوننت‌ها در ری‌اکت را در سایر منابع آموزشی پیدا کنید. می‌توانید این ریپازیتوری گیت‌هاب[۱۲۸] را بررسی کنید تا در مورد راه‌های دیگر برای نوشتن کلاس کامپونت در ری‌اکت بیش‌تر مطالعه کنید.
- **سایر پروژه‌ها:** پس از یادگیری ری‌اکت، همیشه قبل از یادگیری جدید، بهتر است ابتدا آن‌چه را آموخته‌اید در پروژه‌ی خود اعمال کنید. شما می‌توانید در این‌جا آموزش‌های زیادی وجود دارد که از ری‌اکت برای ایجاد چیزی هیجان‌انگیز استفاده می‌کند. در مورد ساخت یک لیست اسکرول بی‌نهایت و صفحه‌بندی[۱۲۹]، نمایش توییت‌ها در صفحه توییتر[۱۳۰] یا اتصال ری‌اکت به stripe برای پرداخت پول[۱۳۱] بیش‌تر بخوانید. تجربه‌ی این اپ‌های کوچک به شما در کار راحت‌تر با ری‌اکت کمک می‌کند.
- **کامپوننت‌های UI:** شما نباید این اشتباه را مرتکب شوید که کتابخانه‌ی کامپوننت UI را خیلی زود در پروژه‌ی خود معرفی کنید. اول باید یاد بگیرید چگونه یک لیست کشویی، چک‌باکس یا دیالوگ را با استفاده از عناصر استاندارد HTML اجرا و استفاده کنید. بخش عمده‌ای از این کامپوننت‌ها state محلی را مدیریت خواهد کرد. یک چک‌باکس باید بداند آیا علامت خورده است یا نه. بنابراین شما باید آن‌ها را به عنوان کامپوننت‌های کنترل‌شده پیاده‌سازی کنید. پس از انجام تمام پیاده‌سازی‌های اولیه، می‌توانید یک کتابخانه‌ی کامپوننت UI معرفی کنید که چک‌باکس‌ها و دیالوگ‌ها را به عنوان کامپوننت‌های ری‌اکت اجرا می‌کند.
- **سازماندهی کد:** در مسیر خواندن این کتاب، شما با یک گفتار در مورد سازماندهی کد روبرو شدید. اگر هنوز این تغییرات را انجام نداده‌اید، اکنون می‌توانید آن را اعمال کنید. این عمل کامپوننت‌های شما را در فایل‌ها و فولدرها (ماژول‌ها)ی ساختاریافته سازماندهی می‌کند. علاوه بر این، به شما کمک می‌کند تا اصول تقسیم کد، استفاده‌ی مجدد، قابلیت نگه‌داری و طراحی API ماژول را درک کرده و یاد بگیرید. در نهایت اپ شما رشد خواهد کرد و شما باید در آن از ماژول‌های ساختاریافته استفاده کنید. پس بهتر است شروع کنید.
- **تست:** این کتاب تنها به تست به‌صورتی بسیار سطحی پرداخته است. اگر شما با اصل موضوع آشنا نیستید، می‌توانید به مفاهیم تست واحد و تست‌های ترکیبی به‌ویژه در زمینه‌ی اپ‌های ری‌اکت عمیق‌تر بپردازید. در سطح اجرا، من توصیه می‌کنم خود را به استفاده از Enzyme و jest محدود کنید تا آزمایش‌های خود را با تست‌های واحد و تست‌های Export در ری‌اکت تکرار کنید.
- **Routing:** شما می‌توانید Routing در اپ خود را با react-router[۱۳۲] انجام دهید. تاکنون، شما فقط یک صفحه در اپ خود داشته اید، react-router به شما کمک می‌کند تا صفحات متعدد را در چندین URL داشته باشید. هنگامی که Routing را در اپ خود معرفی می‌کنید، برای درخواست صفحه‌ی بعدی، هیچ درخواستی به وب سرور نخواهید داشت. router همه چیز را برای شما در سمت مشتری انجام خواهد داد. بنابراین برای معرفی Routing در اپ‌تان شروع به کار کنید.
- **بررسی type :** در یکی از گفتارها شما از React propType برای تعریف تعاملات کامپوننت‌ها استفاده کردید. این یک اقدام کلی خوب برای پیشگیری از باگ‌ها است. اما propTypes فقط در زمان اجرا بررسی می‌شود. شما

[128] https://github.com/the-road-to-learn-react/react-alternative-class-component-syntax
[129] https://www.robinwieruch.de/react-paginated-list/
[130] https://www.robinwieruch.de/react-svg-patterns/
[131] https://www.robinwieruch.de/react-express-stripe-payment/
[132] https://github.com/ReactTraining/react-router

می‌توانید با معرفی روش استاتیک بررسی در زمان کامپایل، یک قدم جلوتر بروید. typeScript[۱۳۳] یک رویکرد محبوب است. اما در اکوسیستم ری‌اکت، مردم اغلب از Flow[۱۳۴] استفاده می کنند. اگر می‌خواهید اپ خود را قوی کنید. توصیه می کنم از Flow استفاده کنید.

- **React Native**: React Native[۱۳۵] اپ شما را در دستگاه های تلفن همراه ایجاد می‌کند. شما می‌توانید آموزش ری‌اکت خود را به سمت اپ‌های اندروید و IOS پیش ببرید. هنگامی که ری‌اکت را یاد می گیرید، منحنی یادگیری در React Native نباید زیاد شیب داشته باشد. هر دو اصول یکسان دارند. شما بیش از مواردی که در اپ های وب استفاده می‌شود، با تفاوت در طرح‌بندی کامپوننت‌ها مواجه خواهید شد.

به طور کلی، از شما دعوت می کنم از «وب‌سایت من[۱۳۶]» برای پیدا کردن موضوعات جالب در مورد توسعه‌ی وب و مهندسی نرم‌افزار بخوانید. شما می‌توانید در «خبرنامه‌ی من[۱۳۷]» برای به‌روزرسانی در مورد مقالات، کتاب‌ها و دوره‌های آموزشی مشترک شوید. علاوه بر این، پلتفرم در مسیر ری‌اکت[۱۳۸] دوره‌های پیشرفته‌تری برای یادگیری اکوسیستم ری‌اکت ارائه می‌دهد. بهتر است آن را هم بررسی کنید.

در نهایت امیدوارم بتوانم اعضای بیش‌تری در Patreon[۱۳۹] پیدا کنم که بتوانند من را در مورد محتوای این کتاب حمایت کنند. دانش‌آموزان و دانشجویان زیادی وجود دارند که نمی‌توانند هزینه‌های آموزشی را پرداخت کنند. به همین دلیل من مقدار زیادی از مطالبم را به صورت رایگان قرار داده‌ام. با حمایت من در این سایت قادر خواهم بود این تلاش‌ها را ادامه دهم تا دیگران به‌صورت رایگان آموزش ببینید.

یک بار دیگر، اگر کتاب را دوست داشتید، می خواهم یک لحظه در مورد افراد دیگری که به‌دنبال یک راه خوب برای یادگیری ری‌اکت هستند فکر کنید. پس این کتاب را با آن‌ها به اشتراک بگذارید. این کار ارزش زیادی برای من دارد. هدف این کتاب هم همین است که با دیگران به اشتراک گذاشته شود. وقتی مردم این کتاب را بیش‌تر بخوانند و نظرات خود را با من به اشتراک بگذارند مطالب آن در طول زمان بهبود خواهد یافت. امیدوارم بازخوردها، بررسی‌ها یا امتیازهای شما را هم ببنیم.

با تشکر از شما برای خواندن «مقدمات یادگیری ری‌اکت».

رابین

133 https://www.typescriptlang.org
134 https://flow.org
135 https://facebook.github.io/react-native/
136 https://www.robinwieruch.de
137 https://www.getrevue.co/profile/rwieruch
138 https://roadtoreact.com
139 https://www.patreon.com/rwieruch