

**El camino para aprender Reaccionar**

Su viaje a dominar sencillo y pragmático Reaccionar

Robin Wieruch

Este libro está a la venta en <http://leanpub.com/the-road-to-learn-react>

Esta versión se publicó el 01/14/2018



Esto es un [Leanpub](http://leanpub.com/) libro. Leanpub permite a los autores y editores con el proceso de publicación de Lean.[editorial magra](http://leanpub.com/manifesto) es el acto de publicar un libro electrónico en curso utilizando herramientas ligeras y muchas iteraciones para obtener retroalimentación de los lectores, de pivote hasta que tenga el libro adecuado y construir la tracción vez que lo hace.

© 2016 - 2018 Robin Wieruch

**Twitter este libro!**

Por favor, ayudar a Robin Wieruch por difundir la información acerca de este libro sobre [Gorjeo](http://twitter.com)!

El tweet sugerido para este libro es:

[Acabo de comprar el camino de aprender reaccionar #ReactJs @rwieruch](https://twitter.com/intent/tweet?text=I%20just%20bought%20The%20Road%20to%20learn%20React%20by%20@rwieruch%20%23ReactJs%20https://roadtoreact.com/course-details?courseId=THE_ROAD_TO_LEARN_REACT) [https://roadtoreact.com/course-details?courseId=THE\_ROAD\_TO\_LEARN\_REACT](https://twitter.com/intent/tweet?text=I%20just%20bought%20The%20Road%20to%20learn%20React%20by%20@rwieruch%20%23ReactJs%20https://roadtoreact.com/course-details?courseId=THE_ROAD_TO_LEARN_REACT)

El hashtag sugerido para este libro es [#ReactJs](https://twitter.com/search?q=%23ReactJs).

Descubra lo que otras personas están diciendo sobre el libro haciendo clic en este enlace para buscar hashtag en Twitter:

[#ReactJs](https://twitter.com/search?q=%23ReactJs)

**Contenido**

|  |  |
| --- | --- |
| **Prefacio** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **yo** |
| **Testimonios** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **ii** |
| **Educación para Niños** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **iv** |
| **Preguntas más frecuentes** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **v** |
| **cambio de registro** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **vii** |
| **Cómo leerlo?** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **ix** |
| **Lo que puede esperar (hasta ahora ...)** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **X** |
| **Introducción a reaccionar** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **1** |
| Hola, mi nombre es React. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| Requisitos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| nodo y MNP. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| Instalación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| Configuración de configuración cero. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| Introducción a JSX. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
| const ES6 y dejar. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
| ReactDOM. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| Reemplazo del módulo caliente. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 18 |
| JavaScript compleja en JSX. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 20 |
| ES6 Flecha Funciones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 24 |
| Clases ES6. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 26 |
| **Conceptos básicos en React** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **29** |
| Estado de componentes internos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 30 |
| ES6 inicializador de objeto. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 33 |
| Flujo de datos unidireccional. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 35 |
| Enlaces. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 40 |
| Controlador de eventos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 45 |

|  |  |
| --- | --- |
| CONTENIDO |  |
| Las interacciones con las formas y Eventos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 50 |
| ES6 desestructuración. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 58 |
| Componentes controlada. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 60 |
| Hasta separación de componentes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 62 |
| Componentes componible. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 66 |
| Componentes reutilizables. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 68 |
| Declaraciones de componentes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 71 |
| Componentes de estilo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 75 |
| **Getting Real con una API** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **82** |
| Métodos de ciclo de vida. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 83 |
| Recuperacion de datos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 86 |
| Los operadores Spread ES6. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 91 |
| Representación condicional. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 94 |
| Del cliente o del lado del servidor de búsqueda. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 97 |
| Fetch paginado. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 101 |
| Caché del cliente. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 105 |
| Manejo de errores . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 112 |
| **Organización de código y pruebas** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **117** |
| Módulos ES6: importación y exportación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 118 |
| Organización código con módulos ES6. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 122 |
| Las pruebas instantáneas con broma. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 127 |
| Las pruebas unitarias con enzima. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 134 |
| La interfaz del componente con PropTypes. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 136 |
| **Reaccionar avanzada Componentes** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **141** |
| Ref un elemento DOM. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 142 |
| Cargando … . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 146 |
| Componentes de orden superior. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 150 |
| Clasificación avanzada. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 154 |
| **Gestión del Estado en reaccionar y más allá** . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **167** |
| Estado de elevación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 168 |
| Revisited: setstate (). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 175 |
| La doma del Estado. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 180 |
| **Los pasos finales para la Producción** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **182** |
| Expulsar. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 183 |
| Despliegue su aplicación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 184 |
| **contorno** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **185** |

**Prefacio**

El camino para aprender Reaccionar le enseña los fundamentos de React. Usted va a construir una aplicación real por el camino en la llanura reaccionar sin herramientas complicadas. Todo, desde la configuración del proyecto para el despliegue en el servidor se explicará. El libro viene acompañado de material de lectura que se hace referencia adicional y ejercicios con cada capítulo. Después de leer el libro, usted será capaz de construir sus propias aplicaciones en React. El material se mantiene hasta la fecha por mí, Robin Wieruch, y la comunidad.

En el camino de aprender Reaccionar, quiero ofrecer una base antes de empezar a sumergirse en el ecosistema más amplio React. Tiene menos utillaje y menos administración de estado externo, pero una gran cantidad de información en torno a reaccionar. Explica los conceptos generales, los patrones y las mejores prácticas en un mundo real Reaccionar aplicación.

Usted aprenderá a construir su propia aplicación React. Cubre características del mundo real como la paginación, la caché del cliente y las interacciones como búsqueda y clasificación. Además te la transición de JavaScript JavaScript ES5 ES6 a lo largo del camino. Espero que este libro capta mi entusiasmo para reaccionar y JavaScript y le ayuda a empezar.

**Testimonios**

[**Muhammad Kashif**](https://twitter.com/appsdevpk/status/848625244956901376)¹**:** “El camino para aprender Reaccionar es un libro único que recomiendo a cualquier estudianteo profesional interesado en el aprendizaje básico reaccionan a nivel avanzado. Está repleto de consejos y técnicas que son difíciles de encontrar en otros lugares, y notablemente a fondo en el uso de ejemplos y referencias a la muestra problemas interesantes, tengo 17 años de experiencia en la web y desarrollo de aplicaciones de escritorio, y antes de leer este libro que estaba teniendo problemas en el aprendizaje reaccionar, pero este libro funciona como magia “.

[**Andre Vargas**](https://twitter.com/andrevar66/status/853789166987038720)²**:** “El camino para aprender Reaccionar por Robin Wieruch es un libro tan impresionante! La mayoría deLo que he aprendido acerca reaccionan e incluso ES6 era a través de él!”

[**Nicholas Hunt-Walker, instructor de Python en una escuela de Codificación Seattle**](https://twitter.com/nhuntwalker/status/845730837823840256)³**:** "Éste es uno demás bien escritos y libros de codificación informativos que he trabajado alguna vez a través. Un sólido Reaccionar y introducción ES6 “.

[**Austin Green**](https://twitter.com/AustinGreen/status/845321540627521536)⁴**:** “Gracias, amado realmente el libro. mezcla perfecta de aprender Reaccionar, ES6, y el nivel más altolos conceptos de programación “.

[**Nicole Ferguson**](https://twitter.com/nicoleffe/status/833488391148822528)⁵**:** “Estoy haciendo camino de Robin para aprender Reaccionar supuesto este fin de semana y casi me sientoculpable por tener tan divertido “.

[**Karan**](https://twitter.com/kvss1992/status/889197346344493056)⁶**:** “Acaba de terminar su camino a reaccionar. Mejor libro para un principiante en el mundo de reaccionar y JS.la exposición elegante de ES. ¡Prestigio! :)”

[**Eric Priou**](https://twitter.com/erixtekila/status/840875459730657283)⁷**:** “Camino a aprender Reaccionar por Robin Wieruch es una lectura obligada. Limpia y concisa para Reaccionary JavaScript “.

**Un novato Desarrollador:** “Acabo de terminar el libro como un desarrollador novato, gracias por trabajar en esto. Era fácil de seguir y me siento confiado en el inicio de una nueva aplicación desde cero en los próximos días. El libro era mucho mejor que el tutorial oficial React.js que he intentado antes (y no podía completar debido a la falta de detalles). Los ejercicios al final de cada sección eran muy gratificante “.

**Estudiante:** “El mejor libro para iniciar ReactJS aprendizaje. El proyecto se mueve junto con los conceptos que se aprendieron que ayuda a comprender el tema. He encontrado 'Código y aprender' como mejor manera de dominar la programación y este libro hace exactamente eso.”

[**Thomas Lockney**](https://www.goodreads.com/review/show/1880673388)⁸**:** “Introducción bastante sólido para reaccionar que no intenta ser exhaustiva. yo soloquería un sabor de entender de qué se trataba y este libro me dio exactamente eso. No seguir todas

¹<https://twitter.com/appsdevpk/status/848625244956901376>

²<https://twitter.com/andrevar66/status/853789166987038720>

³<https://twitter.com/nhuntwalker/status/845730837823840256>

⁴<https://twitter.com/AustinGreen/status/845321540627521536>

⁵<https://twitter.com/nicoleffe/status/833488391148822528>

⁶<https://twitter.com/kvss1992/status/889197346344493056>

⁷<https://twitter.com/erixtekila/status/840875459730657283>

⁸<https://www.goodreads.com/review/show/1880673388>

Testimonios iii

los pequeños pies de página para aprender acerca de las nuevas características ES6 que he perdido ( “Yo no diría que he estado perdiendo él, Bob.”). Pero estoy seguro que para aquellos de ustedes que han quedado atrás y son diligentes en los siguientes, es probable que pueda aprender mucho más que lo que enseña el libro “.

**Educación para Niños**

El libro debe permitir que todos puedan aprender a reaccionar. Sin embargo, no todo el mundo tiene el privilegio de utilizar esos recursos, porque no todo el mundo es educado en el idioma Inglés en el primer lugar. Por lo tanto quiero utilizar el proyecto para apoyar proyectos que enseñan a los niños Inglés en el mundo en desarrollo.

* 1. de abril al 18 de abril de 2017, [Devolviendo, aprendiendo React⁹](https://www.robinwieruch.de/giving-back-by-learning-react/)

⁹<https://www.robinwieruch.de/giving-back-by-learning-react/>

**Preguntas más frecuentes**

**¿Cómo puedo obtener actualizaciones?** Usted puede [subscribe¹⁰](https://www.getrevue.co/profile/rwieruch)a la Newsletter o seguirme en [Twitter¹¹](https://twitter.com/rwieruch)para las actualizaciones. Una vez que tenga una copia del libro, que se mantendrá actualizada cuando una nueva edición se libera. Pero hay que tomar de nuevo la copia cuando se anuncia una actualización.

**¿Utiliza la reciente versión de reaccionar?** El libro siempre recibe una actualización cuando la versión Reaccionar consiguió actualiza. Por lo general, los libros están obsoletos muy pronto después de su liberación. Puesto que este libro es auto-publicado, puedo actualizar cuando quiera.

**¿Cubre Redux?** No lo hace. Por lo tanto, he escrito un segundo libro. El camino para aprender Reaccionar debe darle una base sólida antes de sumergirse en temas avanzados. La implementación de la aplicación de ejemplo en el libro mostrará que no es necesario Redux para construir una aplicación en React. Después de haber leído el libro, usted debe ser capaz de implementar una aplicación sólida y sin Redux. A continuación, puede leer mi segundo libro para aprender[Redux¹²](https://roadtoreact.com/course-details?courseId=TAMING_THE_STATE).

**¿Utiliza JavaScript ES6?** Sí. Pero no se preocupe. Va a estar bien si está familiarizado con el lenguaje Java ES5. Todas las funciones JavaScript ES6, que describo en el viaje de aprender reaccionar, pasará de ES5 a ES6 en el libro. Se explicará todas las características a lo largo del camino. El libro no sólo enseña reaccionar, pero también cuenta con todos útiles JavaScript para ES6 React.

**Va a añadir más capítulos en el futuro?** Puede echar un vistazo al capítulo de Cambio de registro para los principales cambios que ya sucedieron. Habrá mejoras sin previo aviso en el medio también. En general, depende de la comunidad si sigo trabajando en el libro. Si hay una aceptación por el libro, que entregará más capítulos y mejorar el material viejo. Voy a mantener el contenido al día con las últimas mejores prácticas, conceptos y patrones.

**¿Cómo puedo obtener ayuda al leer el libro?** El libro tiene una [Slack Group¹³](https://slack-the-road-to-learn-react.wieruch.com/)para las personas que están leyendo el libro. Usted puede unirse al canal para obtener ayuda o para ayudar a otros. Después de todo, ayudar a los demás puede mejorar sus aprendizajes también.

**¿Hay alguna área de solucionar problemas?** Si se encuentra con problemas, por favor unirse al Grupo Slack. Además, usted podría tener una mirada hacia el[cuestiones abiertas sobre GitHub¹⁴](https://github.com/rwieruch/the-road-to-learn-react/issues) para el libro. Tal vez su problema ya se mencionó y se puede encontrar la solución para ello. Si su problema no fue mencionado, no dude en abrir un nuevo tema en el que puede explicar su problema, tal vez proporcionar una captura de pantalla, y algunos detalles más (por ejemplo, la página del libro, la versión de nodo). Después de todo, trato de enviar todos los arreglos en las próximas ediciones del libro.

¹⁰<https://www.getrevue.co/profile/rwieruch>

¹¹<https://twitter.com/rwieruch>

¹²<https://roadtoreact.com/course-details?courseId=TAMING_THE_STATE>

¹³<https://slack-the-road-to-learn-react.wieruch.com/>

¹⁴<https://github.com/rwieruch/the-road-to-learn-react/issues>

Preguntas más frecuentes vi

**Puedo ayudar a mejorarlo?** Sí. Puede tener un impacto directo con sus pensamientos y[contribuciones](https://github.com/rwieruch/the-road-to-learn-react)[en GitHub¹⁵](https://github.com/rwieruch/the-road-to-learn-react). No pretendo ser un experto ni escribir en Inglés nativo. Le agradecería su ayudará mucho.

**¿Puedo apoyar el proyecto?** Sí. No dude en llegar. Invierto mucho de mi tiempo en tutoriales de código abierto y los recursos de aprendizaje. Puede echar un vistazo a mi[sobre me¹⁶](https://www.robinwieruch.de/about/) página. Me encantaría tenerte como[Patrón de Patreon¹⁷](https://www.patreon.com/rwieruch).

**¿Hay una llamada a la acción?** Sí. Quiero que tome un momento para pensar en una persona que sería un buen partido para aprender a reaccionar. La persona podría haber mostrado el interés ya, podría estar en el medio del aprendizaje reaccionar o no podría ser consciente de que quieran aprender a reaccionar. Llegar a esa persona y compartir el libro. Significaría mucho para mí. El libro está destinado a ser dado a otros.

¹⁵<https://github.com/rwieruch/the-road-to-learn-react>

¹⁶<https://www.robinwieruch.de/about/>

¹⁷<https://www.patreon.com/rwieruch>

**cambio de registro**

1. **De enero de 2017**
   * [v2 Tire Request¹⁸](https://github.com/rwieruch/the-road-to-learn-react/pull/18)
   * aún más amigable para los principiantes
   * 37% más de contenido
   * 30% de contenido mejorado
   * 13 capítulos mejorados y nuevos
   * 140 páginas de material de aprendizaje
   * [+ Curso interactivo del libro sobre educative.io¹⁹](https://www.educative.io/collection/5740745361195008/5676830073815040)
2. **De marzo de 2017**
   * [v3 Tire Request²⁰](https://github.com/rwieruch/the-road-to-learn-react/pull/34)
   * 20% más de contenido
   * 25% de contenido mejorado
   * 9 nuevos capítulos
   * 170 páginas de material de aprendizaje
3. **De abril de 2017**
   * actualizar reaccionar 15,5
4. **De julio de 2017**
   * actualizar a 8.1.3 nodo
   * actualizar a 5.0.4 NPM
   * actualizar para crear una reacción aplicación 1.3.3
5. **De octubre de 2017**
   * actualizar a 8.3.0 nodo

¹⁸<https://github.com/rwieruch/the-road-to-learn-react/pull/18>

¹⁹<https://www.educative.io/collection/5740745361195008/5676830073815040> ²⁰<https://github.com/rwieruch/the-road-to-learn-react/pull/34>

cambio de registro viii

* actualizar a 5.5.1 NPM
* actualizar para crear una reacción aplicación 1.4.1
* actualizar reaccionar 16
* [v4 Tire Request²¹](https://github.com/rwieruch/the-road-to-learn-react/pull/72)
* 15% más de contenido
* 15% de contenido mejorado
* 3 nuevos capítulos (Bindings, controladores de eventos, manejo de errores)
* 190+ páginas de material de aprendizaje
* [9 Código fuente Projects²²](https://roadtoreact.com/course-details?courseId=THE_ROAD_TO_LEARN_REACT)

²¹<https://github.com/rwieruch/the-road-to-learn-react/pull/72>

²²<https://roadtoreact.com/course-details?courseId=THE_ROAD_TO_LEARN_REACT>

**Cómo leerlo?**

El libro es mi intento de enseñar Reaccionar, mientras que usted va a escribir una aplicación. Es una guía práctica para aprender reaccionar y no una obra de referencia sobre React. Usted tendrá que escribir una aplicación de noticias del pirata informático que interactúa con un verdadero API mundo. Entre varios temas interesantes, que abarca la administración del estado en Reaccionar, almacenamiento en caché y las interacciones (clasificación y búsqueda). En el camino que va a aprender las mejores prácticas y patrones en React.

Además, el libro le da una transición de JavaScript ES5 ES6 tener JavaScript. Reaccionar abarca una gran cantidad de características JavaScript ES6 y yo quiero mostrar cómo puede utilizarlos.

En general, cada capítulo del libro se acumulará en el capítulo anterior. Cada capítulo le enseñará algo nuevo. No se apresure a través del libro. Usted debe internalizar cada paso. Se podría aplicar sus propias implementaciones y leer más sobre el tema. Después de cada capítulo te doy un poco de material de lectura y ejercicios. Si usted realmente desea aprender Reaccionar, le recomiendo leer el material extra y hacer algunos ejercicios prácticos. Después de haber leído un capítulo, póngase cómodo con los aprendizajes antes de continuar.

Al final tendrá una completa Reaccionar aplicación en la producción. Estoy muy interesado en ver los resultados, así que por favor me de texto cuando haya terminado el libro. El capítulo final del libro le dará un puñado de opciones para continuar su viaje React. En general se encuentra una gran cantidad de Reaccionar sobre temas relacionados[mi website²³ personales](https://www.robinwieruch.de/).

Puesto que usted está leyendo el libro, supongo que es nuevo para reaccionar. Eso es perfecto. Al final espero recibir sus comentarios para mejorar el material para que todos puedan aprender a reaccionar. Usted puede tener un impacto directo en[GitHub²⁴](https://github.com/rwieruch/the-road-to-learn-react) o texto mí en [Twitter²⁵](https://twitter.com/rwieruch).

²³<https://www.robinwieruch.de/>

²⁴<https://github.com/rwieruch/the-road-to-learn-react>

²⁵<https://twitter.com/rwieruch>

**Lo que puede esperar (hasta ahora ...)**

* [Hacker News App en React²⁶](https://intense-refuge-78753.herokuapp.com/)
* sin configuraciones complicadas
* Create-reaccionar aplicación para arrancar su aplicación
* código ligera eficiente
* sólo reaccionan setstate como la administración del estado (hasta ahora ...)
* transición de JavaScript ES5 ES6 a lo largo del camino
* la Reaccionar API con métodos setstate y de ciclo de vida
* la interacción con un API mundo real (Hacker News)
* interacciones de los usuarios avanzados

**-** clasificación cliente-sided

**-** el filtrado de clientes unilateral

**-** la búsqueda del lado servidor

* implementación de la caché del cliente
* funciones de orden superior y componentes de orden superior
* componentes de la prueba de instantánea con Jest
* componentes de prueba unidad con la enzima
* bibliotecas ordenadas en el camino
* ejercicios y más lecturas en el camino
* interiorizar y reforzar sus aprendizajes
* desplegar su aplicación a la producción

²⁶<https://intense-refuge-78753.herokuapp.com/>

**Introducción a reaccionar**

El capítulo proporciona una introducción a reaccionar. Usted puede preguntarse: ¿Por qué debería aprender reaccionar en primer lugar? El capítulo podría darle la respuesta a esa pregunta. Después, se sumerge en el ecosistema por bootstrapping su primera aplicación Reaccionar desde cero con cero configuración. En el camino, obtendrá una introducción a JSX y ReactDOM. Así que esté preparado para sus primeros componentes reaccionan.

|  |  |
| --- | --- |
| Introducción a reaccionar | 2 |

**Hola, mi nombre es React.**

**¿Por qué te molesta que aprender a reaccionar?** En los últimos años las aplicaciones de una sola página ([SPA²⁷](https://en.wikipedia.org/wiki/Single-page_application)) Han llegado a ser populares. Marcos como angular, Ember y la columna vertebral ayudaron a los desarrolladores de JavaScript para construir aplicaciones web modernas más allá del uso de JavaScript básico y jQuery. La lista de estas soluciones populares no es exhaustiva. Existe una amplia gama de marcos de SPA. Si tenemos en cuenta las fechas de lanzamiento, la mayoría de ellos se encuentran entre la primera generación de SPAs: Angular 2010, Backbone 2010 y 2011 Ember.

El primer lanzamiento fue React 2013 por Facebook. Reaccionar no es un marco de spa, pero una biblioteca de vista. Es la V en el[MVC²⁸](https://de.wikipedia.org/wiki/Model_View_Controller) (Controlador de vista del modelo). Sólo se le permite hacer que los componentes como elementos visibles en un navegador. Sin embargo, todo el ecosistema alrededor Reaccionar hace posible la construcción de aplicaciones de una sola página.

Pero ¿por qué debería considerar el uso Reaccionar sobre la primera generación de marcos de SPA? Mientras que la primera generación de marcos trató de resolver un montón de cosas a la vez, Reaccionar sólo le ayuda a construir su capa de vista. Es una biblioteca y no un marco. La idea detrás de esto: Su punto de vista es una jerarquía de componentes componibles.

En Reaccionar se puede mantener el foco en su capa de vista antes de introducir más aspectos a su aplicación. Todos los demás aspectos es otro bloque de construcción para su spa. Estos bloques de construcción son esenciales para construir una aplicación madura. Vienen con dos ventajas.

En primer lugar, se puede aprender los bloques de construcción paso a paso. Usted no tiene que preocuparse de entender por completo. Es diferente de un marco que le da a cada bloque de construcción desde el principio. Este libro se centra en Reaccionar como el primer bloque de construcción. Más bloques de construcción siguen con el tiempo.

En segundo lugar, todos los bloques de construcción son intercambiables. Esto hace que el ecosistema alrededor Reaccionar un lugar tan innovador. Múltiples soluciones están compitiendo entre sí. Usted puede escoger la solución más atractiva para usted y su caso de uso.

La primera generación de marcos de SPA llegó a un nivel empresarial. Ellos son más rígidos. Reaccionar queda innovador y sea aprobado por múltiples tecnología pensaron empresas líder como[Airbnb, Netflix](https://github.com/facebook/react/wiki/Sites-Using-React) [y por supuesto Facebook²⁹](https://github.com/facebook/react/wiki/Sites-Using-React). Todos ellos invertir en el futuro de reaccionar y se contentan con reaccionar y el ecosistema en sí.

Reaccionar es probablemente una de las mejores opciones para la creación de aplicaciones web modernas hoy en día. Sólo se entrega la capa de vista,[pero el Reaccionar ecosistema es una framework³⁰ flexible y intercambiables toda](https://www.robinwieruch.de/essential-react-libraries-framework/). Reaccionar tiene una API delgada, un ecosistema increíble y una gran comunidad. Usted puede leer acerca de mis experiencias[por eso me mudé de angular a React³¹](https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/). Yo recomiendo tener una comprensión por las que se elige Reaccionar sobre otro marco o en la biblioteca. Después de todo, todo el mundo está dispuesto a experimentar, donde Reaccionar nos llevará en los próximos años.

²⁷<https://en.wikipedia.org/wiki/Single-page_application>

²⁸<https://de.wikipedia.org/wiki/Model_View_Controller>

²⁹<https://github.com/facebook/react/wiki/Sites-Using-React>

³⁰<https://www.robinwieruch.de/essential-react-libraries-framework/>

³¹<https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 3 |

**Ceremonias**

* leer acerca de [por eso me mudé de angular a React³²](https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/)
* leer acerca de [de reaccionar ecosystem³³ flexibles](https://www.robinwieruch.de/essential-react-libraries-framework/)

³²<https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/> ³³<https://www.robinwieruch.de/essential-react-libraries-framework/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 4 |

**requisitos**

Si usted viene de un marco SPA diferente o biblioteca, que ya debe estar familiarizado con los conceptos básicos de desarrollo web. Si usted acaba de comenzar en el desarrollo web, usted debe sentirse cómodo con HTML, CSS y JavaScript ES5 para aprender a reaccionar. El libro será sin problemas la transición a JavaScript ES6 y más allá. Os animo a unirse a la oficial[Slack Group³⁴](https://slack-the-road-to-learn-react.wieruch.com/) para el libro para obtener ayuda o para ayudar a otros.

**Editor y Terminal**

¿Qué pasa con el entorno de desarrollo? Usted necesitará un editor de correr o (herramienta de línea de comandos) IDE y terminal. Usted puede[seguir mi configuración guide³⁵](https://www.robinwieruch.de/developer-setup/). Se ajusta a los usuarios de MacOS, pero se puede sustituir la mayor parte de las herramientas de otro sistema operativo. Hay un montón de artículos hacia fuera allí que le mostrará cómo configurar un entorno de desarrollo web de una manera más elaborada para su sistema operativo.

Opcionalmente, se puede usar Git y Github por su cuenta, mientras que la realización de los ejercicios en el libro, para mantener sus proyectos y los avances en los repositorios de GitHub. Existe una[poco guide³⁶](https://www.robinwieruch.de/git-essential-commands/) sobre el uso de estas herramientas. Pero una vez más, no es obligatorio para el libro y puede ser abrumador al aprender todo desde cero. Así se puede omitir si usted es un recién llegado en el desarrollo web para centrarse en las partes esenciales que se enseñan en este libro.

**Nodo y NPM**

Por último, pero no menos importante, se necesita una instalación de [nodo y npm³⁷](https://nodejs.org/en/). Ambos se utilizan para administrar bibliotecas que va a necesitar en el camino. En este libro, que va a instalar los paquetes de nodos externos a través de la NGP (nodo gestor de paquetes). Estos paquetes de nodos pueden ser bibliotecas enteras o marcos.

Puede verificar sus versiones de nodo y NPM en la línea de comandos. Si usted no recibe ninguna salida en el terminal, es necesario instalar nodo y NPM en primer lugar. Estos son sólo mis versiones durante el tiempo de la escritura de este libro:

**Línea de comando**

--version nodo

\* v8.3.0

NPM --version

\* v5.5.1

³⁴<https://slack-the-road-to-learn-react.wieruch.com/>

³⁵<https://www.robinwieruch.de/developer-setup/>

³⁶<https://www.robinwieruch.de/git-essential-commands/>

³⁷<https://nodejs.org/en/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 5 |

**nodo y NPM**

En este capítulo se le da un poco de curso rápido de nodo y la NGP. No es exhaustiva, pero usted recibirá todas las herramientas necesarias. Si está familiarizado con los dos, puede saltarse este capítulo.

El gestor de paquetes nodo (NPM) le permite instalar paquetes de nodo externas desde la línea de comandos. Estos paquetes pueden ser un conjunto de funciones de utilidad, bibliotecas o marcos enteros. Son las dependencias de su aplicación. Puede instalar estos paquetes a su carpeta de paquete global de nodo o en la carpeta de proyecto local.

paquetes de nodos globales son accesibles desde todas partes en el terminal y hay que instalarlos sólo una vez a su directorio global. Puede instalar un paquete global escribiendo en su terminal:

**Línea de comando**

NPM instalar -g <paquete>

los -gramobandera dice NPM para instalar el paquete global. paquetes locales se utilizan en la aplicación. Por ejemplo, reaccionar como una biblioteca será un paquete local que puede ser requerida en su solicitud de uso. Se puede instalar a través del terminal escribiendo:

**Línea de comando**

NPM instalar <paquete>

En el caso de React sería:

**Línea de comando**

NPM instalar reaccionar

El paquete instalado aparecerá automáticamente en una carpeta llamada node\_modules / y se enumeran en el archivo package.json junto a sus otras dependencias.

Pero la forma de inicializar el node\_modules / carpeta y el archivo package.json para su proyecto en el primer lugar? Hay un comando NPM para inicializar un proyecto NPM y por lo tanto un archivo package.json. Sólo cuando se tiene ese archivo, puede instalar nuevos paquetes locales a través de la NGP.

**Línea de comando**

NPM init -y

|  |  |
| --- | --- |
| Introducción a reaccionar | 6 |

los -ybandera es un acceso directo para inicializar todos los valores predeterminados en su package.json. Si no se utiliza la bandera, usted tiene que decidir cómo configurar el archivo. Después de inicializar su proyecto NPM que son buenos para instalar nuevos paquetes a travésNPM instalar <paquete>.

Una palabra más sobre el package.json. El archivo le permite compartir su proyecto con otros desarrolladores sin compartir todos los paquetes de nodo. El archivo tiene todas las referencias de los paquetes de nodos utilizados en su proyecto. Estos paquetes son llamados dependencias. Todo el mundo puede copiar el proyecto sin las dependencias. Las dependencias son referencias en el package.json. Alguien que copia su proyecto simplemente puede instalar todos los paquetes mediante el uso deNPM instalaren la línea de comandos. losNPM instalar script toma todas las dependencias enumeradas en el *package.json* presentar y los instala en el *node\_modules /* carpeta.

Quiero cubrir una más comandos NPM:

**Línea de comando**

NPM instalar --save-dev <paquete>

los --save-devbandera indica que el paquete nodo sólo se utiliza en el entorno de desarrollo. No va a ser utilizado en la producción cuando se implementa la aplicación en un servidor. ¿Qué tipo de paquete de nodo puede ser? Imagine que desea probar la aplicación con la ayuda de un paquete de nodo. Es necesario instalar ese paquete a través de la NGP, pero quiere excluirlo de su entorno de producción. Las pruebas sólo debería ocurrir durante el proceso de desarrollo, pero no cuando la aplicación se está ejecutando en la producción. Allí no desea probar la aplicación más. Debe ser probado ya y funciona fuera de la caja para sus usuarios. Eso es sólo un caso de uso en el que se desea utilizar el--save-dev bandera.

Usted se encontrará con más comandos de NGP en su camino. Pero estos serán suficientes por ahora.

**Ceremonias:**

* configurar un tiro de proyecto NPM

**-** crear una nueva carpeta con mkdir <nombre\_carpeta>

**-** navegue a la carpeta con cd <nombre\_carpeta>

**-** ejecutar NPM init -yo init NPM

**-** instalar un paquete local, como Reaccionar con NPM instalar reaccionar

**-** echar un vistazo en el archivo package.json y la node\_modules / carpeta

**-** averiguar por su cuenta cómo desinstalar el paquete de nodo de reaccionar de nuevo

* leer más sobre [npm³⁸](https://docs.npmjs.com/)

³⁸<https://docs.npmjs.com/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 7 |

**Instalación**

Hay varios enfoques para empezar con una aplicación React.

La primera de ellas es el uso de un CDN. Esto puede sonar más complicado de lo que es. Un CDN es una[contenido](https://en.wikipedia.org/wiki/Content_delivery_network) [network³⁹ entrega](https://en.wikipedia.org/wiki/Content_delivery_network). Varias compañías tienen CDN que los archivos de host en público por las personas a consumir ellos. Estos archivos pueden ser bibliotecas como Reaccionar, porque después de todo la biblioteca incorporada Reaccionar es sólo un archivo JavaScript react.js. Se puede alojarse en algún lugar y se puede requerir en su aplicación.

Cómo utilizar un CDN para iniciarse en reaccionar? Puede inline la<Script>etiqueta en el código HTML que apunta a una URL de CDN. Para empezar en Reaccionar necesita dos archivos (bibliotecas): reaccionar y reaccionar-dom.

**código de Área de juegos**

<guión crossorigin src="Https://unpkg.com/react@16/umd/react.development.js"> </ \ guión>

<guión crossorigin src="Https: //unpkg.com/react-dom@16/umd/react-dom.developmen \ t.js"> </ Script>

Pero ¿por qué debería utilizar un CDN cuando se tiene NPM para instalar paquetes de nodos como reaccionar?

Cuando su aplicación tiene un archivo package.json, puede instalar reaccionar y reaccionar-dom desde la línea de comandos. El requisito es que la carpeta se inicializa como proyecto mediante el uso de NPMinit NPM -y con un *package.json* archivo. Puede instalar varios paquetes de nodos en una línea con la NGP.

**Línea de comando**

NPM install reaccionar reaccionar-dom

Este enfoque se utiliza a menudo para agregar reaccionar a una aplicación existente que se gestiona con la NGP.

Por desgracia, eso no es todo. Usted tendría que hacer frente a[Babel⁴⁰](http://babeljs.io/) para que su aplicación consciente de JSX (la sintaxis Reaccionar) y JavaScript ES6. Babel transpiles su código para que los navegadores pueden interpretar JavaScript ES6 y JSX. No todos los navegadores son capaces de interpretar la sintaxis. La configuración incluye una gran cantidad de configuración y herramientas. Puede ser abrumador para los recién llegados Reaccionar a molestar con toda la configuración.

Debido a esta razón, Facebook introdujo crear una reacción aplicación como un cero configuración Reaccionar solución.

El siguiente capítulo le mostrará cómo configurar su aplicación mediante el uso de esta herramienta de arranque.

**Ceremonias:**

• leer más sobre [reaccionar installations⁴¹](https://facebook.github.io/react/docs/installation.html)

³⁹<https://en.wikipedia.org/wiki/Content_delivery_network>

⁴⁰<http://babeljs.io/>

⁴¹<https://facebook.github.io/react/docs/installation.html>

|  |  |
| --- | --- |
| Introducción a reaccionar | 8 |

**Configuración de configuración cero**

En el camino de aprender Reaccionar, que va a utilizar [crear una reacción-app⁴²](https://github.com/facebookincubator/create-react-app) para arrancar la aplicación. Se trata de un kit de inicio obstinado aún sin configuración para Reaccionar introducido por Facebook en 2016. La gente se[recomendaría a los principiantes en un 96% ⁴³](https://twitter.com/dan_abramov/status/806985854099062785). En crear una reacción aplicación del utillaje y la configuración evolucionan en el fondo mientras la atención se centra en la implementación de la aplicación.

Para empezar, tendrá que instalar el paquete a los paquetes de nodos globales. Después de eso, siempre tiene a su disposición en la línea de comandos para arrancar nuevas aplicaciones React.

**Línea de comando**

NPM instalar -g crear una reacción aplicación

Puede comprobar la versión de crear una reacción aplicación para verificar la instalación correcta en la línea de comandos:

**Línea de comando**

crear una reacción aplicación --version

\* v1.4.1

Ahora puede arrancar su primera aplicación React. Lo llamamos HackerNews, pero se puede elegir un nombre diferente. El arranque tarda un par de segundos. Después, simplemente navegue a la carpeta:

**Línea de comando**

crear una reacción-app HackerNews

HackerNews cd

Ahora se puede abrir la aplicación en su editor. La siguiente estructura de carpetas, o una variación de la misma en función de la versión de crear una reacción aplicación, se deben presentar a usted:

⁴²<https://github.com/facebookincubator/create-react-app>

⁴³<https://twitter.com/dan_abramov/status/806985854099062785>

|  |  |
| --- | --- |
| Introducción a reaccionar | 9 |

**Estructura de carpetas**

HackerNews /

README.md

node\_modules /

package.json

.gitignore

público/

favicon.ico

index.html

manifest.json

src /

App.css

app.js

App.test.js

index.css

index.js

logo.svg

registerServiceWorker.js

Un breve descanso abajo de la carpeta y los archivos. Está bien si usted no entiende todos ellos en el principio.

* **README.md:** La extensión .md indica que el archivo es un archivo de rebajas. Reducción del precio se utiliza como lenguaje de marcado ligero, con sintaxis de formato de texto plano. Muchos proyectos de código fuente vienen con un archivo README.md para darle instrucciones iniciales sobre el proyecto. Al empujar su proyecto para una plataforma como GitHub, finalmente, el archivo README.md mostrará su contenido prominentemente cuando se accede al repositorio. Debido a que usted ha utilizado crear reaccionan aplicación, sus README.md debe ser el mismo que se muestra en el funcionario[crear una reacción aplicación GitHub](https://github.com/facebookincubator/create-react-app)[repository⁴⁴](https://github.com/facebookincubator/create-react-app).
* **node\_modules /:** La carpeta tiene todos los paquetes de nodos que fueron y se instalan a través de la NGP. Puesto que usted ha utilizado crear reaccionan aplicación, no debe haber ya un par de módulos de nodos instalados para usted. Por lo general, nunca se toca esta carpeta, pero sólo instalar y desinstalar paquetes de nodos con la NGP desde la línea de comandos.
* **package.json:** El archivo que muestra una lista de dependencias de paquetes nodo y otra configuración del proyecto.
* **.gitignore:** El archivo indica todos los archivos y carpetas que no deben añadirse a su repositorio Git remoto cuando se utiliza git. Sólo deben vivir en su proyecto local. El node\_modules / carpeta es un caso tal uso. Es suficiente para compartir el archivo package.json con sus pares para que puedan instalar todas las dependencias de su propia sin compartir la carpeta completa dependencia.

⁴⁴<https://github.com/facebookincubator/create-react-app>

|  |  |
| --- | --- |
| Introducción a reaccionar | 10 |

* **público/:** La carpeta contiene todos los archivos en la construcción de su proyecto para la producción. Con el tiempo todo el código escrito en la carpeta / src se incluirá en un par de archivos cuando la construcción de su proyecto y se coloca en la carpeta pública.
* **manifest.json** y registerServiceWorker.js: no se preocupan por lo que hacen estos archivos en esta etapa, no vamos a ellos que necesitan en este proyecto.

Después de todo, no es necesario tocar los archivos y las carpetas mencionadas. Al principio todo lo que necesita se encuentra en la carpeta / src. El foco principal se encuentra en el archivo / src app.js para implementar Reaccionar componentes. Será utilizado para implementar su aplicación, pero más tarde es posible que desee dividir sus componentes en varios archivos mientras que mantiene cada archivo de uno o unos pocos componentes por sí solo.

Además, encontrará un archivo / src App.test.js para sus pruebas y unos src / index.js como punto de entrada al mundo React. Se llega a conocer ambos archivos en un capítulo posterior. Además, hay un src / index.css y un archivo / src App.css el estilo de su aplicación general y sus componentes. Todas están equipadas con estilo por defecto cuando los abra.

La aplicación crear una reacción aplicación es un proyecto de la NGP. Puede utilizar la NGP para instalar y desinstalar paquetes de nodos a su proyecto. Además viene con las siguientes secuencias de comandos de la NGP para su línea de comandos:

**Línea de comando**

* Ejecuta la aplicación en http: // localhost: 3000 inicio NPM
* Ejecuta las pruebas

prueba de NPM

* Construye la solicitud de NPM producción de carreras de construcción

Las secuencias de comandos se definen en su package.json. Su aplicación es repetitivo Reaccionar bootstrap ahora.

La parte interesante viene en los ejercicios para ejecutar finalmente su aplicación bootstrap en el navegador.

**Ceremonias:**

* inicio NPM su aplicación y visitar la aplicación en su navegador (se puede salir de la comando presionando Control + C)
* ejecutar el interactivo prueba de NPM guión
* ejecutar el NPM plazo de construcción guión y verificar que un / carpeta de compilación ha sido añadido a su proyecto (se puede quitar de nuevo después; tenga en cuenta que la carpeta de compilación se puede utilizar más adelante para [desplegar su](https://www.robinwieruch.de/deploy-applications-digital-ocean/) [application⁴⁵](https://www.robinwieruch.de/deploy-applications-digital-ocean/))

⁴⁵<https://www.robinwieruch.de/deploy-applications-digital-ocean/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 11 |

* familiarizarse con la estructura de carpetas
* familiarizarse con el contenido de los archivos
* leer más sobre [los guiones de la NGP y crear una reacción-app⁴⁶](https://github.com/facebookincubator/create-react-app)

⁴⁶<https://github.com/facebookincubator/create-react-app>

|  |  |
| --- | --- |
| Introducción a reaccionar | 12 |

**Introducción a JSX**

Ahora se va a conocer JSX. Es la sintaxis en React. Como se mencionó antes, crear una reacción aplicación ya ha bootstrap una aplicación de texto modelo para usted. Todos los archivos vienen con implementaciones por defecto. Vamos a bucear en el código fuente. El único archivo que va a tocar en un principio será el archivo / app.js src.

**src / app.js**

**importar** Reaccionar, {} Componente de'reaccionar';

**importar** logotipo de'./logo.svg';

**importar** ' ./App.css';

**clase** Aplicación **se extiende** Componente {

render () {

**regreso** (

<div className="Aplicación">

<cabecera className="App-header">

<img src={Logo} className="App-logotipo" alt="logo" /> <h1 className="App-título">Bienvenido a reaccionar</ H1>

</ Header>

<p className="App-intro">

Para empezar, editar <código>src/app.js</ Code> y guardar para recargar.

</ P>

</ Div>

);

}

}

**por defecto de exportación** Aplicación;

No se deje conseguir confundido por las declaraciones de importación / exportación y declaración de la clase. Estas características ya están JavaScript ES6. Vamos a volver a los de un capítulo posterior.

En el archivo que tiene un componente de clase ES6 reaccionar con el nombre de la aplicación. Es una declaración de componentes. Básicamente después de haber declarado un componente, se puede utilizar como elemento de todas partes en su aplicación. Se producirá una instancia de su componente o dicho de otro modo: el componente se crea una instancia.

El elemento que devuelve se especifica en el hacer()método. Los elementos son los componentes que están hechos. Es útil para entender las diferencias entre componentes, instancia y elemento.

Muy pronto, se verá en el que se crea una instancia del componente de aplicación. De lo contrario, no podría ver el resultado representado en el navegador, ¿verdad? El componente de aplicación es únicamente la declaración, pero no el uso. Se podría crear una instancia del componente en algún lugar de su JSX con<App />.

|  |  |
| --- | --- |
| Introducción a reaccionar | 13 |

El contenido en el bloque de render se ve muy similar al HTML, pero es JSX. JSX le permite mezclar HTML y JavaScript. Es potente y confuso cuando estás acostumbrado a separar el código HTML y JavaScript. Es por eso que un buen punto de partida es usar HTML básico en su JSX. En un principio, eliminar todo el contenido de distracción en el archivo.

**src / app.js**

**importar** Reaccionar, {} Componente de'reaccionar';

**importar** ' ./App.css';

**clase** Aplicación **se extiende** Componente {

render () {

**regreso** (

<div className="Aplicación">

<h2>Bienvenido a la Ruta de aprender Reaccionar</ H2> </ Div>

);

}

}

**por defecto de exportación** Aplicación;

Ahora, sólo regresa HTML en su hacer()método sin JavaScript. Vamos a definir el “Bienvenido a la carretera para aprender React” como una variable. Una variable puede ser utilizado en su JSX mediante el uso de llaves.

**src / app.js**

Reaccionar importación, {} Componente de 'reaccionar';

importación' ./App.css';

clase App extiende Componente {

render () {

**var helloWorld = 'Bienvenido a la Carretera de aprender React';**

regreso (

<Div className = "App">

**<h2> {helloWorld} </ h2>**

</ Div>

);

}

}

Aplicación por defecto de exportación;

|  |  |
| --- | --- |
| Introducción a reaccionar | 14 |

Debería funcionar cuando se inicia el programa desde la línea de comandos con inicio NPM de nuevo.

Además es posible que haya notado la nombre de la claseatributo. Refleja la normaclaseatribuir en HTML. Debido a razones técnicas, JSX tuvo que sustituir a un puñado de atributos HTML internas. Puede encontrar toda la[apoyado atributos HTML en el Reaccionar documentation⁴⁷](https://facebook.github.io/react/docs/dom-elements.html). Todos siguen la convención camelCase. En su manera de aprender reaccionar, se encontrará con algunos atributos específicos más JSX.

**Ceremonias:**

* definir más variables y hacerlos en su JSX

**-** utilizar un objeto complejo para representar a un usuario con un nombre de pila y apellido

**-** hacer que las propiedades de usuario en su JSX

* leer más sobre [JSX⁴⁸](https://facebook.github.io/react/docs/introducing-jsx.html)
* leer más sobre [Reaccionar componentes, elementos y instances⁴⁹](https://facebook.github.io/react/blog/2015/12/18/react-components-elements-and-instances.html)

⁴⁷<https://facebook.github.io/react/docs/dom-elements.html>

⁴⁸<https://facebook.github.io/react/docs/introducing-jsx.html>

⁴⁹<https://facebook.github.io/react/blog/2015/12/18/react-components-elements-and-instances.html>

|  |  |
| --- | --- |
| Introducción a reaccionar | 15 |

**ES6 const y dejar**

Creo que se ha notado que declaramos la variable Hola Mundo con un vardeclaración. JavaScript ES6 viene con dos opciones más para declarar sus variables:const y dejar. En JavaScript ES6, que rara vez se encuentravar nunca más.

Una variable declarada con constno puede ser reasignado o re-declarado. No puede ser mutado (cambiado, modificado). Abrazas estructuras de datos inmutables mediante el uso de la misma. Una vez definida la estructura de datos, no se puede cambiar.

**código de Área de juegos**

*// No permitido*

**const** Hola Mundo='Bienvenido a la Carretera de aprender React';Hola Mundo = 'Bye Bye React';

Una variable declarada con dejar pueden quedar mutado.

**código de Área de juegos**

*// permitido*

**dejar** Hola Mundo='Bienvenido a la Carretera de aprender React';Hola Mundo = 'Bye Bye React';

Se podría usar cuando se tendría que volver a asignar una variable.

Sin embargo, hay que tener cuidado con const. Una variable declarada conconstno puede quedar modificado. Pero cuando la variable es una matriz o un objeto, el valor que posee puede se actualiza. El valor que posee no es inmutable.

**código de Área de juegos**

*// permitido*

**const** Hola Mundo={

texto: 'Bienvenido a la Carretera de aprender React'

};

helloWorld.text = 'Bye Bye React';

Pero cuando usar cada declaración? Hay diferentes opiniones sobre el uso. Se sugiere emplearconst cuando puedas. Indica que desea mantener la estructura de datos inmutables incluso aunque los valores de los objetos y las matrices pueden quedar modificado. Si desea modificar la variable, puede utilizardejar.

Inmutabilidad es abrazado en reaccionar y su ecosistema. Es por esoconstdebe ser su opción por defecto cuando se define una variable. Aun así, en objetos complejos los valores dentro pueden quedar modificado. Tenga cuidado acerca de este comportamiento.

En su aplicación, se debe utilizar const encima var.

|  |  |
| --- | --- |
| Introducción a reaccionar | dieciséis |

**src / app.js**

Reaccionar importación, {} Componente de 'reaccionar';

importación' ./App.css';

clase App extiende Componente {

render () {

**const helloWorld = 'Bienvenido a la Carretera de aprender React';** regreso (

<Div className = "App">

<h2> {helloWorld} </ h2>

</ Div>

);

}

}

Aplicación por defecto de exportación;

**Ceremonias:**

* leer más sobre [const⁵⁰ ES6](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)
* leer más sobre [let⁵¹ ES6](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let)
* investigar más sobre las estructuras de datos inmutables

**-** ¿por qué tienen sentido en la programación en general

**-** por qué se utilizan en reaccionar y su ecosistema

⁵⁰<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const> ⁵¹<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

|  |  |
| --- | --- |
| Introducción a reaccionar | 17 |

**ReactDOM**

Antes de continuar con el componente de la aplicación, es posible que desee para ver donde se utiliza. Se encuentra en su punto de entrada al mundo React: el archivo src / index.js.

**src / index.js**

**importar** reaccionar de'reaccionar';

**importar** ReactDOM de'Reaccionar-dom';

**importar** Aplicación de'./App';

**importar** ' ./index.css';

ReactDOM.render (

<Aplicación />,

documento.getElementById ('raíz')

);

Básicamente ReactDOM.render ()utiliza un nodo DOM en el código HTML para reemplazarlo con su JSX. Así es como se puede integrar fácilmente reaccionar de cada solicitud extranjera. No está prohibido el usoReactDOM.render () varias veces a través de su aplicación. Se puede utilizar en múltiples lugares a bootstrap sintaxis sencilla JSX, un componente Reaccionar, múltiples componentes reaccionan o toda una aplicación. Sin embargo, en la aplicación Reaccionar llanura va a utilizar sólo una vez para arrancar el árbol de componentes conjunto.

ReactDOM.render () espera dos argumentos. El primer argumento es que consigue JSX prestados. El segundo argumento especifica el lugar donde el Reaccionar ganchos de aplicación en tu HTML. Se espera que un elemento con unaid = 'root'. Puede abrir su archivo público / index.html para encontrar el atributo id.

En la puesta en práctica ReactDOM.render ()Ya tiene su componente de aplicaciones. Sin embargo, sería muy bien para pasar más simple JSX siempre que es JSX. No tiene por qué ser una instancia de un componente.

**código de Área de juegos**

ReactDOM.render (

<h1>Hola Reaccionar Mundial</ H1>,

documento.getElementById ('raíz')

);

**Ceremonias:**

* abrir al público / index.html para ver donde el Reaccionar aplicaciones engancha en su HTML
* leer más sobre [elementos de representación en React⁵²](https://facebook.github.io/react/docs/rendering-elements.html)

⁵²<https://facebook.github.io/react/docs/rendering-elements.html>

|  |  |
| --- | --- |
| Introducción a reaccionar | 18 |

**Sustitución módulo hot**

Hay una cosa que se puede hacer en el archivo index.js src / para mejorar su experiencia de desarrollo como desarrollador. Pero es opcional y no debe abrumar al principio, cuando el aprendizaje de reaccionar.

En crear una reacción aplicación ya es una ventaja que el navegador se actualiza automáticamente la página cuando cambie su código fuente. Pruebe cambiando elHola Mundovariable en el archivo / app.js src. El navegador debe actualizar la página. Pero hay una mejor forma de hacerlo.

Reemplazo del módulo caliente (HMR) es una herramienta para recargar su aplicación en el navegador. El navegador no realiza una actualización de la página. Se puede activar fácilmente en crear una reacción aplicación. En sus src / index.js, su punto de entrada para reaccionar, hay que añadir una pequeña configuración.

**src / index.js**

importación Reaccionar de 'reaccionar';

ReactDOM importación de 'reaccionar-dom';

Aplicación de importación './App';

importación' ./index.css';

ReactDOM.render (

<App />,

document.getElementById ( 'root')

);

**si (module.hot) {**

**module.hot.accept ();**

**}**

Eso es. Inténtelo de nuevo para cambiar laHola Mundovariable en el archivo / app.js src. El navegador no debe llevar a cabo una actualización de la página, pero vuelve a cargar la aplicación y muestra la salida correcta. HMR viene con múltiples ventajas:

Imagínese que usted está depurando el código con console.log ()declaraciones. Estas declaraciones se quedarán en su consola de desarrollador, a pesar de que cambie su código, ya que el navegador no se actualiza la página más. Eso puede ser conveniente para los propósitos de depuración.

En una aplicación cada vez mayor de una actualización de la página retrasa su productividad. Usted tiene que esperar a que se cargue la página. Una recarga de la página puede tardar varios segundos en una aplicación de gran tamaño. HMR quita esta desventaja.

El mayor beneficio es que se puede mantener el estado de la aplicación con HMR. Imagine que tiene un cuadro de diálogo en su aplicación con múltiples pasos y estás en el paso 3. Básicamente se trata de un asistente. Sin HMR que iba a cambiar el código fuente de su navegador y actualiza la página. Usted tendría que abrir el cuadro de diálogo de nuevo y sin tener que navegar desde el paso 1 al paso 3. Con su HMR de diálogo permanece abierto

|  |  |
| --- | --- |
| Introducción a reaccionar | 19 |

en el paso 3. Se mantiene el estado de la aplicación a pesar de que el código fuente cambia. La aplicación en sí vuelve a cargar, pero no la página.

**Ceremonias:**

* cambiar el código de src / fuente app.js un par de veces para ver HMR en la acción
* ver los primeros 10 minutos de [Reaccionar vivir: Recarga caliente con Time Travel⁵³](https://www.youtube.com/watch?v=xsSnOQynTHs) por Dan Abramov

⁵³<https://www.youtube.com/watch?v=xsSnOQynTHs>

|  |  |
| --- | --- |
| Introducción a reaccionar | 20 |

**JavaScript compleja en JSX**

Volvamos a su componente de aplicación. Hasta el momento que renderizó algunas variables primitivas en su JSX. Ahora usted comenzará a hacer una lista de elementos. La lista estará datos de ejemplo en el principio, pero más tarde se obtendrá los datos de una externa[API⁵⁴](https://www.robinwieruch.de/what-is-an-api-javascript/). Eso será mucho más excitante.

En primer lugar tiene que definir la lista de elementos.

**src / app.js**

Reaccionar importación, {} Componente de 'reaccionar';

importación' ./App.css';

**lista const = [**

**{**

**Título: 'reaccionar',**

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

**Autor: Jordan Walke ',**

**NUM\_COMMENTS: 3,**

**puntos: 4,**

**IDobjeto: 0,**

**},**

**{**

**Título: 'Redux',**

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

**Autor: Dan Abramov, Andrew Clark,**

**NUM\_COMMENTS: 2,**

**puntos: 5,**

**IDobjeto: 1,**

**},**

**];**

clase App extiende Componente {

...

}

Los datos de la muestra se reflejan los datos vamos a ir a buscar más tarde de la API. Un elemento de la lista tiene un título, una URL y un autor. Además viene con un identificador, puntos (que indican qué tan popular es un artículo) y un recuento de los comentarios.

Ahora se puede utilizar el incorporado en JavaScript mapafuncionalidad en su JSX. Le permite iterar sobre la lista de elementos para mostrarlos. Una vez más va a utilizar llaves para encapsular la expresión JavaScript en su JSX.

⁵⁴<https://www.robinwieruch.de/what-is-an-api-javascript/>

|  |  |
| --- | --- |
| Introducción a reaccionar | 21 |

**src / app.js**

clase App extiende Componente {

render () {

regreso (

<Div className = "App">

**{List.map (función (elemento) {**

**volver <div> {item.title} </ div>;**

**})}**

</ Div>

);

}

}

Aplicación por defecto de exportación;

El uso de JavaScript en HTML es muy poderoso en JSX. Por lo general, es posible que haya utilizadomapapara convertir una lista de elementos a otra lista de elementos. Esta vez que utilicemapa para convertir una lista de elementos a los elementos HTML.

Hasta el momento, sólo el títulose mostrará para cada elemento. Vamos a mostrar un poco más de las propiedades de los elementos.

**src / app.js**

clase App extiende Componente {

render () {

regreso (

<Div className = "App">

**{List.map (función (elemento) {**

**regreso (**

**<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>

);

}

|  |  |
| --- | --- |
| Introducción a reaccionar | 22 |

}

Aplicación por defecto de exportación;

Se puede ver cómo la función de mapa es simplemente inlined en su JSX. Cada propiedad del elemento se muestra en una<Span>etiqueta. Por otra parte la propiedad url del elemento se utiliza en elhref atributo de la etiqueta de anclaje.

Reaccionar hará todo el trabajo por usted y mostrar cada elemento. Sin embargo, se debe añadir un ayudante para reaccionar a abrazar todo su potencial y mejorar su rendimiento. Tiene que asignar un atributo clave para cada elemento de la lista. De esa manera Reaccionar es capaz de identificar añadido, cambiado y artículos cuando la lista de cambios eliminado. Los elementos de la lista de muestras vienen con un identificador ya.

**src / app.js**

{List.map (función (elemento) {

regreso (

**<Div clave = {} 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>

);

})}

Usted debe asegurarse de que el atributo clave es un identificador estable. No cometa el error de utilizar el índice del elemento de la matriz. El índice de matriz no es estable en absoluto. Por ejemplo, cuando la lista cambia su orden, Reaccionar tendrá un tiempo difícil la identificación de los elementos correctamente.

**src / app.js**

*// no hacer esto*

{List.map (**función**(Elemento, clave) {

**regreso** (

<div clave={llave}>

...

</ Div>

);

})}

Se está mostrando ambos elementos de la lista ahora. Usted puede comenzar su aplicación, abra el navegador y ver tanto los elementos de la lista mostrada.

|  |  |
| --- | --- |
| Introducción a reaccionar | 23 |

**Ceremonias:**

* leer más sobre [Reaccionar listas y keys⁵⁵](https://facebook.github.io/react/docs/lists-and-keys.html)
* recapitular el [funcionalidades incorporadas en la matriz estándar en JavaScript⁵⁶](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map)
* utilizar expresiones de JavaScript más por su cuenta en JSX

⁵⁵<https://facebook.github.io/react/docs/lists-and-keys.html>

⁵⁶<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map>

|  |  |
| --- | --- |
| Introducción a reaccionar | 24 |

**ES6 Flecha Funciones**

JavaScript ES6 introdujo las funciones de dirección. Una expresión de la función de flecha es más corta que una expresión de función.

**código de Área de juegos**

* *expresión de función* **función** () {...}
* *expresión de la función de flecha* () => {...}

Pero hay que ser conscientes de sus funcionalidades. Uno de ellos es un comportamiento diferente con elestaobjeto. Una expresión de función siempre define su propioestaobjeto. las expresiones de función Flecha todavía tienen laestaobjeto del contexto de encerramiento. No se confunda al utilizaresta en una función de la flecha.

Hay otro hecho valiosa acerca de las funciones de dirección en relación con los paréntesis. Puede eliminar los paréntesis cuando la función obtiene sólo un argumento, sino que tienen que mantenerlos cuando se pone múltiples argumentos.

**código de Área de juegos**

*// permitido*

item => {...}

*// permitido*

(Punto) => {...}

*// No permitido*

elemento, clave => {...}

*// permitido*

(Elemento, clave) => {...}

Sin embargo, vamos a echar un vistazo a la mapafunción. Se puede escribir de manera más concisa con una función de flecha ES6.

|  |  |
| --- | --- |
| Introducción a reaccionar | 25 |

**src / app.js**

**{List.map (item => {**

regreso (

<Div clave = {} 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>

);

})}

Además, se puede quitar el cuerpo de bloque, es decir, las llaves, la función de flecha ES6. En un cuerpo concisa rendimiento, implícito se adjunta. De este modo se puede quitar la instrucción de retorno. Eso va a ocurrir con más frecuencia en el libro, así que asegúrese de entender la diferencia entre un cuerpo y un cuerpo de bloque concisa al utilizar las funciones de dirección.

**src / app.js**

**{List.map (item =>**

<Div clave = {} 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>

**)}**

Su JSX parece más concisa y fácil de leer ahora. Se omite la declaración de la función, las llaves y la instrucción de retorno. En cambio, un desarrollador puede concentrarse en los detalles de implementación.

**Ceremonias:**

• leer más sobre [ES6 flecha functions⁵⁷](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions)

⁵⁷<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

|  |  |
| --- | --- |
| Introducción a reaccionar | 26 |

**Clases ES6**

JavaScript ES6 introdujo clases. Una clase se utiliza comúnmente en los lenguajes de programación orientados a objetos. JavaScript ha sido y es muy flexible en sus paradigmas de programación. Usted puede hacer la programación funcional y el lado de programación orientado a objetos al lado del otro por sus casos de uso particulares.

A pesar de que Reaccionar abarca la programación funcional, por ejemplo, con las estructuras de datos inmutables, las clases se utilizan para declarar componentes. Se llaman los componentes de clase ES6. Reaccionar mezcla las partes buenas de ambos paradigmas de programación.

Vamos a considerar la siguiente clase de desarrollador para examinar una clase de JavaScript ES6 sin pensar en un componente.

**código de Área de juegos**

**clase** Desarrollador {

constructor (nombre, apellido) {

**esta**.nombre de pila=nombre de pila;

**esta**.apellido=apellido;

}

getName () {

**devolver este**.nombre de pila+''+ **esta**.apellido;

}

}

Una clase tiene un constructor para que sea instanciable. El constructor puede tomar argumentos para asignarlo a la instancia de clase. Además una clase puede definir funciones. Dado que la función está asociada con una clase, se llama un método. A menudo se hace referencia como un método de clase.

La clase Developer es solamente la declaración de clase. Se pueden crear múltiples instancias de la clase mediante la invocación de la misma. Es similar al componente de clase ES6, que tiene una declaración, pero hay que usarlo en otro lugar para crear instancias de ella.

Vamos a ver cómo se puede crear una instancia de la clase y cómo se puede utilizar sus métodos.

**código de Área de juegos**

**const** Robin= **nuevo** Desarrollador('Robin','Wieruch');console.log (robin.getName ());

*// salida: Robin Wieruch*

Reaccionar utiliza clases JavaScript ES6 para los componentes de clase ES6. Ya ha utilizado un componente de clase ES6.

|  |  |
| --- | --- |
| Introducción a reaccionar | 27 |

**src / app.js**

**importar** Reaccionar, {} Componente de'reaccionar';

...

**clase** Aplicación **se extiende** Componente {

render () {

...

}

}

La clase App extiende desde Componente. Básicamente se declara el componente de aplicación, sino que se extiende a partir de otro componente. Lo que se extienden significa? En la programación orientada a objetos que tiene el principio de la herencia. Se utiliza para pasar por encima de las funcionalidades de una clase a otra clase.

La clase App amplía la funcionalidad de la clase de componentes. Para ser más específicos, hereda las funcionalidades de la clase Component. La clase de componente se utiliza para extender una clase básica ES6 a una clase de componente ES6. Tiene todas las funcionalidades que un componente en Reaccionar necesidades de tener. El método render es una de estas funcionalidades que ya se ha utilizado. Usted aprenderá acerca de otros métodos de clases de componentes más adelante.

los Componenteclase encapsula todos los detalles de implementación de un componente React. Se permite a los desarrolladores usar clases como componentes en React.

Los métodos de un Reaccionar Componenteexpone es la interfaz pública. Uno de estos métodos tiene que ser anulado, los otros no necesitan ser anulado. Usted aprenderá acerca de estas últimas cuando el libro llega a métodos de ciclo de vida en un capítulo posterior. loshacer() método tiene que ser anulado, ya que define la salida de un Reaccionar Componente. Tiene que ser definido.

Ahora que sabes lo básico alrededor de JavaScript clases ES6 y la forma en que se utilizan en reaccionar a extenderlas a los componentes. Va a aprender más acerca de los métodos de componentes cuando el libro se describen Reaccionar métodos de ciclo de vida.

**Ceremonias:**

• leer más sobre [classes⁵⁸ ES6](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes)

⁵⁸<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes>

|  |  |
| --- | --- |
| Introducción a reaccionar | 28 |

Has aprendido a arrancar su propio Reaccionar aplicación! Recapitulemos los últimos capítulos:

* Reaccionar

**-** Create-reaccionan-app bootstraps una aplicación Reaccionar

**-** JSX se mezcla HTML y JavaScript para definir la salida de componentes reaccionan en sus métodos de render

**-** componentes, instancias y elementos son cosas diferentes en React

**-** ReactDOM.render ()es un punto de entrada para una aplicación Reaccionar para enganchar Reaccionar en el DOM

**-** incorporado en JavaScript funcionalidades se pueden utilizar en JSX

\* Mapa se puede utilizar para hacer una lista de elementos como elementos HTML

* ES6

**-** con declaraciones de variables consty dejarpuede ser utilizado para casos de uso específicos

* + utilizar const sobre aplicaciones permiten la entrada de React

**-** funciones de dirección se pueden utilizar para mantener sus funciones concisa

**-** clases se utilizan para definir los componentes en React extendiéndolas

Tiene sentido hacer una pausa en este punto. Internalizar los aprendizajes y aplicarlos en su propio.

Usted puede experimentar con el código fuente que ha escrito hasta ahora.

Puede encontrar el código fuente en el [repository⁵⁹ oficial](https://github.com/rwieruch/hackernews-client/tree/4.1).

⁵⁹<https://github.com/rwieruch/hackernews-client/tree/4.1>

**Conceptos básicos en React**

El capítulo le guiará a través de los fundamentos de React. Cubre el estado y las interacciones de los componentes, ya que los componentes estáticos son un poco aburrido, ¿no es así? Además, usted aprenderá acerca de las diferentes formas de declarar un componente y cómo mantener los componentes componibles y reutilizable. Esté preparado para dar vida a sus componentes.

|  |  |
| --- | --- |
| Conceptos básicos en React | 30 |

**Estado de componentes internos**

el estado del componente interno, también conocido como estado local, le permite guardar, modificar y borrar propiedades que se almacenan en su componente. El componente de clase ES6 puede utilizar un constructor para inicializar el estado del componente interno más adelante. El constructor se llama sólo una vez cuando el componente inicializa.

Vamos a introducir un constructor de la clase.

**src / app.js**

clase App extiende Componente {

**constructor (apoyos) {**

**super (apoyos);**

**}**

...

}

Al tener un constructor en su componente de clase ES6, es obligatorio para llamar súper(); porque el componente de aplicación es una subclase de Componente. Por lo tanto, laComponente extiendeen su declaración de componentes de aplicaciones. Va a aprender más acerca de los componentes de clase ES6 más adelante.

Puedes llamar super (apoyos);también. En él se establecenthis.propsen su constructor en caso de que quiera acceder a ellos en el constructor. De lo contrario, cuando se accedethis.props en su constructor, que serían indefinido. Va a aprender más acerca de los puntales de un componente Reaccionar más adelante.

Ahora, en su caso, el estado inicial en su componente debe ser la lista de muestras de artículos.

**src / app.js**

lista const = [

{

Título: 'reaccionar',

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

Autor: Jordan Walke ',

NUM\_COMMENTS: 3,

puntos: 4,

IDobjeto: 0,

},

...

];

|  |  |
| --- | --- |
| Conceptos básicos en React | 31 |

clase App extiende Componente {

constructor (apoyos) {

super (apoyos);

**this.state = {**

**Lista: Lista,**

**};**

}

...

}

El Estado está obligado a la clase mediante el uso de la estaobjeto. De este modo se puede acceder al estado local en todo su componente. Por ejemplo, se puede utilizar en elhacer()método. Anteriormente ha asignado una lista estática de artículos en suhacer()método que se define fuera de su componente. Ahora que está a punto de utilizar la lista de su estado local en su componente.

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

**{This.state.list.map (item =>**

<Div clave = {} 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>

);

}

}

|  |  |
| --- | --- |
| Conceptos básicos en React | 32 |

La lista es parte del componente de ahora. Reside en el estado del componente interno. Se podría añadir objetos, cambiar los elementos o eliminar elementos en y desde su lista. Cada vez que cambie su estado de componente, elhacer()método de su componente se ejecutará de nuevo. Eso es cómo se puede simplemente cambiar de estado de los componentes internos y estar seguro de que el componente vuelve a renderizar y muestra los datos correctos que viene del estado local.

Pero ten cuidado. No mutar el estado directamente. Usted tiene que utilizar un método llamadosetstate ()para modificar su estado. Se llega a saber que en un capítulo siguiente.

**Ceremonias:**

* experimento con el estado local

**-** definir el estado más inicial en el constructor

**-** usar y acceder al estado en su hacer()método

* leer más sobre [constructor⁶⁰ la clase ES6](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes#Constructor)

⁶⁰<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes#Constructor>

|  |  |
| --- | --- |
| Conceptos básicos en React | 33 |

**Inicializador de objeto ES6**

En JavaScript ES6, puede utilizar una sintaxis propiedad abreviada para inicializar los objetos de manera más concisa.

Imagine la siguiente inicialización de objetos:

**código de Área de juegos**

**const** nombre='Robin';

**const** usuario={

nombre: nombre,

};

Cuando el nombre de la propiedad en su objeto es el mismo que el nombre de la variable, puede hacer lo siguiente:

**código de Área de juegos**

**const** nombre='Robin';

**const** usuario={

nombre,

};

En su aplicación, puede hacer lo mismo. La lista de nombres de variable y el nombre de la propiedad del Estado comparten el mismo nombre.

**código de Área de juegos**

*// ES5*

**esta**.estado={

lista: lista,

};

*// ES6*

**esta**.estado={

lista,

};

Otro ayudante ordenada son los nombres de métodos abreviados. En JavaScript ES6, puede inicializar los métodos de un objeto de forma más concisa.

|  |  |
| --- | --- |
| Conceptos básicos en React | 34 |

**código de Área de juegos**

*// ES5*

**var** UserService={

getUserName: **función** (Usuario) {

**regreso** user.firstname+''+user.lastname;

},

};

*// ES6*

**const** UserService={

getUserName (usuario) {

**regreso** user.firstname+''+user.lastname;

},

};

Por último, pero no menos importante, se le permite utilizar nombres de las propiedades calculadas en JavaScript ES6.

**código de Área de juegos**

*// ES5*

**var** usuario={

nombre: 'Robin',

};

*// ES6*

**const** llave='nombre';

**const** usuario={

[llave]: 'Robin',

};

Tal vez calculados los nombres de propiedades no tienen sentido para usted todavía. ¿Por qué si los necesita? En un capítulo posterior, se llega a un punto en el que se pueden utilizar para asignar los valores de clave de forma dinámica en un objeto. Es ordenado para generar tablas de búsqueda en JavaScript.

**Ceremonias:**

* experimento con ES6 objeto inicializador
* leer más sobre [ES6 objeto initializer⁶¹](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Object_initializer)

⁶¹<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Object_initializer>

|  |  |
| --- | --- |
| Conceptos básicos en React | 35 |

**Flujo de datos unidireccional**

Ahora usted tiene algún estado interno en el componente de aplicación. Sin embargo, no se ha manipulado el estado local todavía. El estado es estática y por lo tanto es el componente. Una buena manera de experimentar la manipulación del estado es tener algún tipo de interacción de los componentes.

Vamos a añadir un botón para cada elemento de la lista que se muestra. El botón dice “Descartar” y va a eliminar el elemento de la lista. Podría ser útil, finalmente, cuando sólo se desea mantener una lista de elementos no leídos y separar a los elementos que no está interesada.

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

{This.state.list.map (item =>

<Div clave = {} 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>**

**<botón**

**onclick = {() => this.onDismiss (item.objectID)} type = "button"**

**>**

**Despedir**

**</ Botón>**

**</ Span>**

</ Div>

)}

</ Div>

);

}

}

los onDismiss ()método de clase no está definido todavía. Lo haremos en un momento, pero por ahora el foco debe estar en elal hacer clicmanejador del elemento de botón. Como se puede ver, elonDismiss () método

|  |  |
| --- | --- |
| Conceptos básicos en React | 36 |

en el al hacer clicmanejador está encerrado por otra función. Es una función de la flecha. De esa manera, puede colarse en elIDobjeto propiedad de la ítoponerse a identificar el elemento que va a ser despedido. Una forma alternativa sería definir el exterior en función de laal hacer clicmanejador y sólo pase la función definida al controlador. Un capítulo más adelante se explicará el tema de los manipuladores de elementos con más detalle.

¿Se fijaron las líneas múltiples para el elemento de botón? Tenga en cuenta que los elementos con múltiples atributos se ponen complicadas como una línea en algún momento. Es por eso que el elemento de botón se utiliza con líneas múltiples y hendiduras que siga siendo legible. Pero no es obligatorio. Es sólo una recomendación estilo de código que recomiendo encarecidamente.

Ahora usted tiene que poner en práctica el onDismiss ()funcionalidad. Se necesita un identificador para identificar el elemento de descartar. La función está ligada a la clase y por lo tanto se convierte en un método de clase. Es por eso que acceda a ella conthis.onDismiss () y no onDismiss (). losestaobjeto es la instancia de la clase. Con el fin de definir elonDismiss ()como método de clase, usted tiene que unirse en el constructor. Encuadernaciones se explicarán en otro capítulo más adelante.

**src / app.js**

clase App extiende Componente {

constructor (apoyos) {

super (apoyos);

this.state = {

lista,

};

**this.onDismiss = this.onDismiss.bind (this);**

}

render () {

...

}

}

En el siguiente paso, usted tiene que definir su funcionalidad, la lógica de negocio, en su clase. Los métodos de clase se pueden definir de la siguiente manera.

|  |  |
| --- | --- |
| Conceptos básicos en React | 37 |

**src / app.js**

clase App extiende Componente {

constructor (apoyos) {

super (apoyos);

this.state = {

lista,

};

this.onDismiss = this.onDismiss.bind (this);

}

**onDismiss (id) {**

**...**

**}**

render () {

...

}

}

Ahora usted es capaz de definir lo que ocurre en el interior del método de clase. Básicamente desea quitar el elemento identificado por el ID de la lista y almacenar una lista actualizada a su estado local. Después, la lista actualizada se utiliza en la re-ejecuciónhacer()Método para visualizarlo. El elemento eliminado no debe aparecer nunca más.

Puede eliminar un elemento de una lista utilizando la funcionalidad de filtro incorporado en el código JavaScript. La función de filtro toma una función como entrada. La función tiene acceso a cada valor de la lista, ya que itera sobre la lista. De esta manera, se puede evaluar cada elemento de la lista en base a una condición de filtro. Si la evaluación de un artículo es true, el elemento se queda en la lista. De lo contrario, será filtrada de la lista. Además, es bueno saber que la función devuelve una nueva lista y no muta la lista de edad. Es compatible con la convención en Reaccionar de tener estructuras de datos inmutables.

|  |  |
| --- | --- |
| Conceptos básicos en React | 38 |

**src / app.js**

onDismiss (id) {

**const = updatedList this.state.list.filter (función isNotId (elemento) {return item.objectID id ==!;**

**});**

}

En el siguiente paso, se puede extraer la función y pasarlo a la función de filtro.

**src / app.js**

onDismiss (id) {

**isNotId función (elemento) {**

**volver item.objectID id ==!;**

**}**

**const updatedList = this.state.list.filter (isNotId);**

}

Además, puede hacerlo de forma más concisa mediante el uso de una función de JavaScript flecha ES6 nuevo.

**src / app.js**

onDismiss (id) {

**const isNotId = item => item.objectID id ==!;**

**const updatedList = this.state.list.filter (isNotId);**

}

Incluso se puede inline de nuevo, como lo hizo en el al hacer clic manejador del botón, pero podría obtener menos legible.

**src / app.js**

onDismiss (id) {

**const = updatedList this.state.list.filter (! item => item.objectID == id);**

}

La lista elimina el elemento se ha hecho clic ahora. Sin embargo, el estado no se actualiza todavía. Por lo tanto, que finalmente puede utilizar elsetstate () método de clase para actualizar la lista en el estado de los componentes internos.

|  |  |
| --- | --- |
| Conceptos básicos en React | 39 |

**src / app.js**

onDismiss (id) {

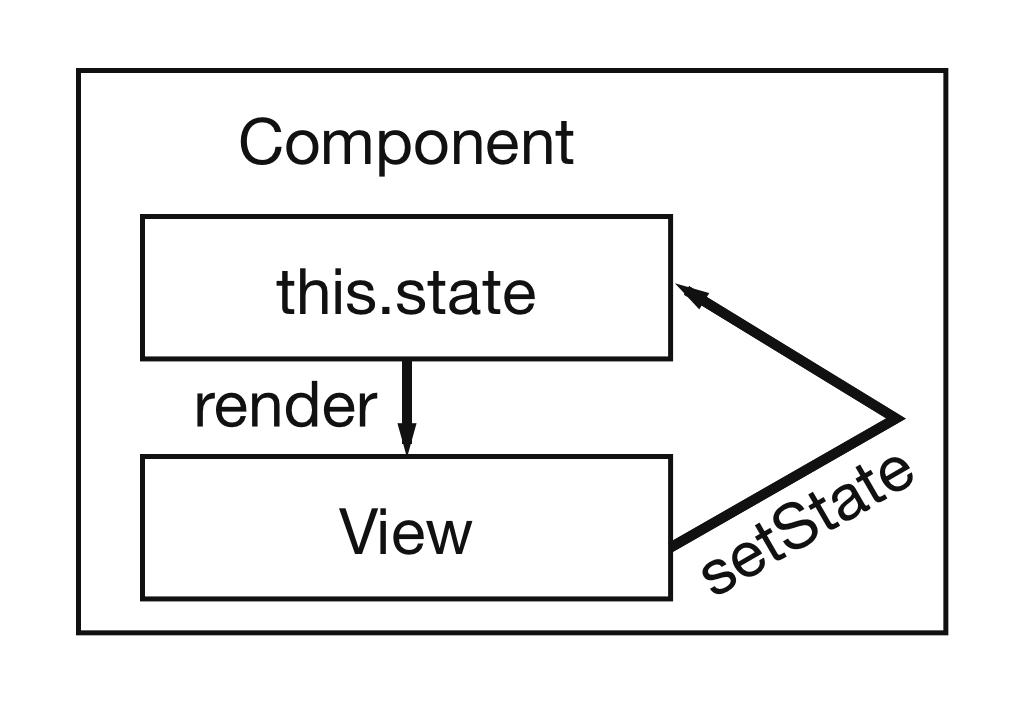
const isNotId = item => item.objectID id ==!;

const updatedList = this.state.list.filter (isNotId);

**this.setState ({list: updatedList});**

}

Ahora ejecute de nuevo la aplicación y probar el botón “Descartar”. Deberia de funcionar. Lo que experimentas ahora es el flujo de datos unidireccional en React. Se dispara una acción en su opinión conal hacer clic(), Un método de función o clase modifica el estado de los componentes internos y el hacer() método del componente se ejecuta de nuevo para actualizar la vista.



**actualización de estado interno con el flujo de datos unidireccional**

**Ceremonias:**

• leer más sobre [el estado del ciclo de vida y en React⁶²](https://facebook.github.io/react/docs/state-and-lifecycle.html)

⁶²<https://facebook.github.io/react/docs/state-and-lifecycle.html>

|  |  |
| --- | --- |
| Conceptos básicos en React | 40 |

**encuadernaciones**

Es importante aprender acerca de las consolidaciones en las clases de JavaScript cuando se utiliza Reaccionar ES6 componentes de clase.

En el capítulo anterior, que ha obligado a su método de clase onDismiss () en el constructor.

**src / app.js**

**clase** Aplicación **se extiende** Componente {

constructor (apoyos) {

**súper**(accesorios);

**esta**.estado={

lista,

};

**esta**.onDismiss= **esta**.onDismiss.bind (**esta**);

}

...

}

¿Por qué haces eso en primer lugar? La etapa de unión es necesario, porque los métodos de clase no se unen de forma automáticaestaa la instancia de la clase. Vamos a demostrar que con la ayuda de los siguientes componentes de clase ES6.

**código de Área de juegos**

**clase** ExplainBindingsComponent **se extiende** Componente {onClickMe () {

console.log (**esta**);

}

render () {

**regreso** (

<botón

al hacer clic={**esta**.onClickMe}

tipo="botón"

>

Haz click en mi

</ Botón>

);

}

}

|  |  |
| --- | --- |
| Conceptos básicos en React | 41 |

El componente rinde muy bien, pero cuando se hace clic en el botón, obtendrá indefinidoen su registro de la consola de desarrolladores. Esa es la principal fuente de errores cuando se utiliza Reaccionar, ya que si se desea accedereste estado en su método de clase, no puede ser recuperada porque esta es indefinido. Así que con el fin de hacer esta accesibles en métodos de su clase, tiene que obligar a los métodos de la clase esta.

En el componente de clase siguiendo el método de clase está ligado correctamente en el constructor de la clase.

**código de Área de juegos**

clase ExplainBindingsComponent extiende Componente {

**constructor () {**

**súper();**

**this.onClickMe = this.onClickMe.bind (this);**

**}**

onClickMe () {

console.log (this);

}

render () {

regreso (

<botón

onClick = {} this.onClickMe

type = "botón"

>

Haz click en mi

</ Botón>

);

}

}

Al tratar de nuevo, el botón esta objeto, para ser más específicos de la instancia de clase, debe ser definida y que sería capaz de acceder este estado, O como más tarde aprenderá this.props, ahora.

El método de la clase de unión que puede suceder en otro lugar también. Por ejemplo, puede suceder en elhacer() método de clase.

|  |  |
| --- | --- |
| Conceptos básicos en React | 42 |

**código de Área de juegos**

clase ExplainBindingsComponent extiende Componente {onClickMe () {

console.log (this);

}

render () {

regreso (

<botón

**onClick = {this.onClickMe.bind (esto)}**

type = "botón"

>

Haz click en mi

</ Botón>

);

}

}

Sin embargo, se debe evitar, ya que se uniría al método de clase cada vez que el hacer()método se ejecuta. Básicamente se ejecuta cada vez que sus actualizaciones de componentes que conduce a consecuencias en el rendimiento. Cuando el enlace de método de clase en el constructor, que ates sólo una vez al principio, cuando el componente se crea una instancia. Eso es un mejor enfoque para hacerlo.

Otra cosa que la gente a veces ocurre es definir la lógica de negocio de sus métodos de clase en el constructor.

**código de Área de juegos**

clase ExplainBindingsComponent extiende Componente {constructor () {

súper();

**this.onClickMe = () => {**

**console.log (this);**

**}**

}

render () {

regreso (

<botón

onClick = {} this.onClickMe

type = "botón"

|  |  |
| --- | --- |
| Conceptos básicos en React | 43 |

>

Haz click en mi

</ Botón>

);

}

}

Se debe evitar también, ya que el desorden de su constructor con el tiempo. El constructor sólo está allí para crear una instancia de la clase con todas sus propiedades. Es por eso que la lógica empresarial de métodos de clase debe definirse fuera del constructor.

**código de Área de juegos**

**clase** ExplainBindingsComponent **se extiende** Componente {constructor () {

**súper**();

**esta**.hacer algo= **esta**.doSomething.bind (**esta**);

**esta**.hacer algo más= **esta**.doSomethingElse.bind (**esta**);

}

hacer algo() {

*// hacer algo*

}

hacer algo más() {

*// hacer algo más*

}

...

}

Por último, pero no menos importante, vale la pena mencionar que los métodos de la clase pueden ser autobound automáticamente sin unirlos de forma explícita mediante el uso de funciones de dirección JavaScript ES6.

|  |  |
| --- | --- |
| Conceptos básicos en React | 44 |

**código de Área de juegos**

**clase** ExplainBindingsComponent **se extiende** Componente {onClickMe = () => {

console.log (**esta**);

}

render () {

**regreso** (

<botón

al hacer clic={**esta**.onClickMe}

tipo="botón"

>

Haz click en mi

</ Botón>

);

}

}

Si la unión en el constructor repetitivo le molesta, puede seguir adelante con este enfoque en su lugar. El funcionario Reaccionar palos de documentación a los enlaces de método de clase en el constructor. Es por eso que el libro se adherirá a esos también.

**Ceremonias:**

• tratar los diferentes enfoques de fijaciones y consolar a registrar el esta objeto

|  |  |
| --- | --- |
| Conceptos básicos en React | 45 |

**Controlador de eventos**

El capítulo debe darle una comprensión más profunda de los controladores de eventos de elementos. En su aplicación, se utiliza el siguiente elemento de botón para despedir a un elemento de la lista.

**src / app.js**

...

<botón

al hacer clic={() => **esta**.onDismiss (item.objectID)} Tipo="botón"

>

Despedir

</ Botón>

...

Que ya es un caso de uso complejo, ya que tiene que pasar un valor al método de la clase y por lo tanto hay que envolverlo en otra función (flecha). Así que, básicamente, tiene que ser una función que se pasa al controlador de eventos. El siguiente código no funcionaría, ya que el método de la clase sería ejecutado inmediatamente cuando se abre la aplicación en el navegador.

**src / app.js**

...

<botón

al hacer clic={**esta**.onDismiss (item.objectID)}

tipo="botón"

>

Despedir

</ Botón>

...

Cuando usas onClick = {doSomething ()}, el hacer algo()función ejecutaría inmediatamente cuando se abre la aplicación en su navegador. La expresión en el controlador se evalúa. Dado que el valor devuelto de la función no es una función más, no pasaría nada al hacer clic en el botón. Sin embargo, cuando se usaonClick = {} doSomething mientras hacer algoes una función, que sería ejecutado cuando se hace clic en el botón. Las mismas reglas se aplican para laonDismiss () método de la clase que se utiliza en la aplicación.

|  |  |
| --- | --- |
| Conceptos básicos en React | 46 |

Sin embargo, el uso de onClick = {} this.onDismiss no sería suficiente, porque de alguna manera la item.objectIDpropiedad tiene que ser pasado al método de clase para identificar el elemento que va a ser despedido. Es por eso que puede ser envuelto en otra función de colarse en la propiedad. El concepto se llama funciones de orden superior en JavaScript y se explicará brevemente más adelante.

**src / app.js**

...

<botón

al hacer clic={() => **esta**.onDismiss (item.objectID)} Tipo="botón"

>

Despedir

</ Botón>

...

Una solución sería definir la función de envolver en algún lugar fuera y sólo pasar a la función definida al controlador. Ya que necesita acceso al elemento individual, tiene que vivir en el interior del bloque de función de mapa.

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

{This.state.list.map (item => {

**const onHandleDismiss = () =>**

**this.onDismiss (item.objectID);**

regreso (

<Div clave = {} 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>

|  |  |
| --- | --- |
| Conceptos básicos en React | 47 |

<botón

**onClick = {} onHandleDismiss**

type = "botón"

>

Despedir

</ Botón>

</ Span>

</ Div>

);

}

)}

</ Div>

);

}

}

Después de todo, tiene que ser una función que se pasa al manejador del elemento. A modo de ejemplo, probar este código en su lugar:

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

{This.state.list.map (item =>

...

<Span>

<botón

**onClick = {console.log (item.objectID)}**

type = "botón"

>

Despedir

</ Botón>

</ Span>

</ Div>

)}

</ Div>

);

|  |  |
| --- | --- |
| Conceptos básicos en React | 48 |

}

}

Se ejecutará cuando se abre la aplicación en el navegador, pero no cuando se hace clic en el botón. Mientras que el siguiente código sólo se ejecuta cuando hace clic en el botón. Es una función que se ejecuta cuando se dispara el manejador.

**src / app.js**

...

<botón

**onClick = {function () {**

**console.log (item.objectID)**

**}}**

type = "botón"

>

Despedir

</ Botón>

...

Con el fin de mantenerlo concisa, puede transformarla en una función JavaScript flecha ES6 nuevo. Eso es lo que hicimos con elonDismiss () método de clase también.

**src / app.js**

...

<botón

**onclick = {() => console.log (item.objectID)}**

type = "botón"

>

Despedir

</ Botón>

...

A menudo, los recién llegados a reaccionar tener dificultades con el tema del uso de las funciones de los controladores de eventos. Es por eso que he intentado explicar con más detalle aquí. Al final, usted debe terminar con el siguiente código en su botón para tener una función de JavaScript flecha ES6 concisa inline que tenga acceso a laIDobjeto propiedad de la ít objeto.

|  |  |
| --- | --- |
| Conceptos básicos en React | 49 |

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

{This.state.list.map (item =>

<Div clave = {} item.objectID>

...

<Span>

**<botón**

**onclick = {() => this.onDismiss (item.objectID)} type = "button"**

**>**

**Despedir**

**</ Botón>**

</ Span>

</ Div>

)}

</ Div>

);

}

}

Otro tema relevante rendimiento, que se menciona a menudo, son las implicaciones del uso de las funciones de dirección en los controladores de eventos. Por ejemplo, laal hacer clic controlador para el onDismiss ()método está envolviendo el método en otra función de flecha para ser capaz de pasar el identificador de elemento. Así que cada vez que elhacer()método se ejecuta, el controlador crea la instancia de la función de flecha de orden superior. Puede tener un impacto en su rendimiento de las aplicaciones, pero en la mayoría de los casos no lo notará. Imagínese que tiene una gran tabla de datos con 1000 artículos y cada fila o columna tiene una función tan flecha en un controlador de eventos. Entonces vale la pena pensar en las implicaciones de rendimiento y por lo tanto se podría implementar un componente de botón dedicado para enlazar el método en el constructor. Pero antes de que eso ocurra es la optimización prematura. Es más valioso para centrarse en el aprendizaje Reaccionar en sí.

**Ceremonias:**

• tratar los diferentes enfoques del uso de funciones en el al hacer clic manejador de su botón

|  |  |
| --- | --- |
| Conceptos básicos en React | 50 |

**Las interacciones con las formas y Eventos**

Vamos a añadir otra interacción para la aplicación de las formas de experiencia y eventos en React. La interacción es una funcionalidad de búsqueda. La entrada del campo de búsqueda se debe utilizar para filtrar temporalmente la lista basada en la propiedad del título de un artículo.

En el primer paso, se va a definir un formulario con un campo de entrada en su JSX.

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

**<Form>**

**<Input type = "text" />**

**</ Form>**

{This.state.list.map (item =>

...

)}

</ Div>

);

}

}

En el siguiente escenario que se escriba en el campo de entrada y filtrar la lista temporalmente por el término de búsqueda que se utiliza en el campo de entrada. Para ser capaz de filtrar la lista basándose en el valor del campo de entrada, es necesario almacenar el valor del campo de entrada en su estado local. Pero, ¿cómo se accede al valor? Puede usar los eventos sintéticos en React para acceder a la carga de evento.

Vamos a definir una onChange controlador para el campo de entrada.

|  |  |
| --- | --- |
| Conceptos básicos en React | 51 |

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

<Form>

**<input**

**type = "text"**

**onChange = {} this.onSearchChange**

**/>**

</ Form>

...

</ Div>

);

}

}

La función está ligada a la componente y por lo tanto un método de clase de nuevo. Usted tiene que unirse y definir el método.

**src / app.js**

clase App extiende Componente {

constructor (apoyos) {

super (apoyos);

this.state = {

lista,

};

**this.onSearchChange = this.onSearchChange.bind (this);** this.onDismiss = this.onDismiss.bind (this);

}

**onSearchChange () {**

**...**

**}**

|  |  |
| --- | --- |
| Conceptos básicos en React | 52 |

...

}

Cuando se utiliza un controlador en su elemento, se obtiene acceso al sintética Reaccionar evento en la firma de su función de devolución de llamada.

**src / app.js**

clase App extiende Componente {

...

**onSearchChange (evento) {**

...

}

...

}

El evento tiene el valor del campo de entrada en su objeto de destino. Por lo tanto usted es capaz de actualizar el estado local con el término de búsqueda mediante el uso dethis.setState () de nuevo.

**src / app.js**

clase App extiende Componente {

...

onSearchChange (evento) {

**this.setState ({searchTerm: event.target.value});**

}

...

}

Además, no hay que olvidar para definir el estado inicial para el término de búsquedapropiedad en el constructor. El campo de entrada debe estar vacío en el principio y por lo tanto el valor debe ser una cadena vacía.

|  |  |
| --- | --- |
| Conceptos básicos en React | 53 |

**src / app.js**

clase App extiende Componente {

constructor (apoyos) {

super (apoyos);

this.state = {

lista,

**término de búsqueda: '',**

};

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

}

...

}

Ahora se almacena el valor de entrada a su estado componente interno cada vez que el valor de los cambios en el campo de entrada.

Una breve nota sobre la actualización del estado local en un componente React. Sería razonable suponer que al actualizar eltérmino de búsqueda con this.setState ()la lista debe ser aprobado también para preservarlo. Pero ese no es el caso. de reaccionarthis.setState ()es una fusión superficial. Conserva las propiedades de hermanos en el objeto de estado cuando se actualiza una propiedad exclusiva en ella. Así, el estado lista, a pesar de que ya se ha descartado un elemento de ella, permanecería igual al actualizar eltérmino de búsqueda propiedad.

Volvamos a la aplicación. La lista no está filtrada todavía basa en el valor del campo de entrada que se almacena en el estado local. Básicamente, usted tiene que filtrar la lista basada temporalmente en eltérmino de búsqueda. Usted tiene todo lo necesario para filtrarla. Entonces, ¿cómo filtrar temporalmente ahora? En tushacer()método, antes de asignar más de la lista, se puede aplicar un filtro en él. El filtro sólo evaluaría si eltérmino de búsquedapartidos propiedad título del artículo. Ya ha utilizado la funcionalidad del filtro incorporado en JavaScript, así que vamos a hacerlo de nuevo. Puede colarse en la función de filtro antes de la función de mapa, debido a que la función de filtro devuelve una nueva matriz y por lo tanto la función de mapa se puede utilizar en él de tal manera conveniente.

|  |  |
| --- | --- |
| Conceptos básicos en React | 54 |

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

<Form>

<input

type = "text"

onChange = {} this.onSearchChange

/>

</ Form>

**mapa {this.state.list.filter (...). (item =>**

...

)}

</ Div>

);

}

}

Vamos a acercarse a la función de filtro de una manera diferente esta vez. Queremos definir el argumento de filtro, la función que se pasa a la función de filtro, fuera del componente de clase ES6. Hay que no tienen acceso al estado del componente y por lo tanto no tenemos acceso a latérmino de búsquedapropiedad para evaluar la condición del filtro. Tenemos que pasar eltérmino de búsquedaa la función de filtro y tiene que devolver una nueva función para evaluar la condición. Eso se llama una función de orden superior.

Normalmente no me mencionar funciones de orden superior, pero en un libro reaccionan tiene sentido total. Tiene sentido para saber acerca de las funciones de orden superior, porque Reaccionar trata de un concepto llamado componentes de orden superior. Se llega a conocer el concepto más adelante en el libro. Ahora, de nuevo, vamos a centrarnos en la funcionalidad del filtro.

En primer lugar, usted tiene que definir la función de orden superior fuera de su componente de aplicación.

|  |  |
| --- | --- |
| Conceptos básicos en React | 55 |

**src / app.js**

**función isSearched (searchTerm) {**

**función de retorno (elemento) {**

**// alguna condición que devuelve verdadero o falso**

**}**

**}**

clase App extiende Componente {

...

}

La función toma el término de búsqueday devuelve otra función, porque después de todo la función de filtro toma una función como su entrada. La función devuelta tiene acceso al objeto tema, ya que es la función que se pasa a la función de filtro. Además, la función devuelto se utiliza para filtrar la lista basándose en la condición definida en la función. Vamos a definir la condición.

**src / app.js**

función isSearched (searchTerm) {

función de retorno (elemento) {

**regreso item.title.toLowerCase () incluye (searchTerm.toLowerCase ()).;**

}

}

clase App extiende Componente {

...

}

La condición dice que coloca el entrante término de búsquedapatrón con la propiedad título del elemento de la lista. Usted puede hacer eso con el built-inincluyela funcionalidad de JavaScript. Sólo cuando el patrón coincide, se vuelve verdadera y el elemento queda en la lista. Cuando el patrón no coincide con el elemento se elimina de la lista. Pero tenga cuidado con la coincidencia de patrones: No hay que olvidar a minúsculas ambas cadenas. De lo contrario habrá desajustes entre un término de búsqueda 'redux' y un título del artículo 'Redux'. Dado que estamos trabajando en una lista inmutable y devolver una nueva lista utilizando la función de filtro, la lista original en el estado local no se modifica en absoluto.

Una cosa que se deja de mencionar: Hicimos trampa un poco utilizando el built-in incluye la funcionalidad de JavaScript. Ya es una característica ES6. ¿Cómo se vería eso en JavaScript ES5? Se podría utilizar elíndice de()función para obtener el índice del elemento de la lista. Cuando el artículo esté en la lista,índice de() devolverá su índice en la matriz.

|  |  |
| --- | --- |
| Conceptos básicos en React | 56 |

**código de Área de juegos**

* *ES5* String.indexOf (patrón) ! == -1
* *ES6*

string.includes (patrón)

Otra refactorización ordenada se puede hacer con una función de flecha ES6 nuevo. Esto hace que la función más concisa:

**código de Área de juegos**

*// ES5*

**función** isSearched (searchTerm) {

**función de retorno**(ít) {

**regreso** item.title.toLowerCase () incluye (searchTerm.toLowerCase ()).;

}

}

*// ES6*

**const** isSearched=searchTerm => item =>item.title.toLowerCase () incluye (searchTerm.toLowerCase ()).;

Se podría argumentar que funcionan es más fácil de leer. Personalmente me quedo con la segunda. El ecosistema Reaccionar utiliza una gran cantidad de conceptos de programación funcional. Sucede con frecuencia que va a utilizar una función que devuelve una función (funciones de orden superior). En JavaScript ES6, se puede expresar estos más concisa con funciones de dirección.

Por último, pero no menos importante, usted tiene que utilizar el definido isSearched ()función para filtrar la lista. Se le pasa latérmino de búsquedapropiedad de su estado local, devuelve la función de entrada del filtro, y filtra la lista en función de la condición de filtro. Después se asigna sobre la lista filtrada para mostrar un elemento para cada elemento de la lista.

|  |  |
| --- | --- |
| Conceptos básicos en React | 57 |

**src / app.js**

clase App extiende Componente {

...

render () {

regreso (

<Div className = "App">

<Form>

<input

type = "text"

onChange = {} this.onSearchChange

/>

</ Form>

**{This.state.list.filter (isSearched (this.state.searchTerm)). Mapa (item =>**

...

)}

</ Div>

);

}

}

La funcionalidad de búsqueda debería funcionar ahora. Inténtelo usted mismo en el navegador.

**Ceremonias:**

* leer más sobre [reaccionar events⁶³](https://facebook.github.io/react/docs/handling-events.html)
* leer más sobre [functions⁶⁴ de orden superior](https://en.wikipedia.org/wiki/Higher-order_function)

⁶³<https://facebook.github.io/react/docs/handling-events.html>

⁶⁴<https://en.wikipedia.org/wiki/Higher-order_function>

|  |  |
| --- | --- |
| Conceptos básicos en React | 58 |

**La desestructuración ES6**

Hay una forma en JavaScript ES6 para un acceso más fácil a las propiedades de los objetos y las matrices. Se llama desestructuración. Compare el siguiente fragmento de JavaScript ES5 y ES6.

**código de Área de juegos**

**const** usuario={

nombre de pila: 'Robin',

apellido: 'Wieruch',

};

*// ES5*

**var** nombre de pila=user.firstname;

**var** apellido=user.lastname;

console.log (nombre + '' + apellido);

* *Salida: Robin Wieruch*
* *ES6*

**const** { nombre Apellido }=usuario;

console.log (nombre + '' + apellido);

*// salida: Robin Wieruch*

Si bien hay que añadir una línea adicional cada vez que desee acceder a una propiedad de objeto en JavaScript ES5, puede hacerlo en una sola línea en JavaScript ES6. Una buena práctica para facilitar la lectura es utilizar líneas múltiples cuando desestructurar un objeto en varias propiedades.

**código de Área de juegos**

**const** {

nombre de pila,

apellido

} = usuario;

Lo mismo ocurre con las matrices. Puede desestructurar ellos también. Una vez más, las líneas múltiples mantendrán su código susceptible de ser analizada y legible.

|  |  |
| --- | --- |
| Conceptos básicos en React | 59 |

**código de Área de juegos**

**const** usuarios=['Robin','Andrés','Dan'];

**const** [

userOne,

userTwo,

userThree

* = usuarios;

console.log (userOne, userTwo, userThree);

*// salida: Robin Andrew Dan*

Tal vez usted ha notado que el objeto de estado local en el componente de aplicación puede obtener desestructurado de la misma manera. Puede acortar la línea de filtros y mapa de código.

**src / app.js**

render () {

**const {searchTerm, lista} = this.state;**

regreso (

<Div className = "App">

...

**{List.filter (isSearched (searchTerm)). Mapa (item =>**

...

)}

</ Div>

);

Puede hacerlo de la manera ES5 ES6 o:

**código de Área de juegos**

*// ES5*

**var** término de búsqueda= **esta**.state.searchTerm;

**var** lista= **esta**.state.list;

*// ES6*

**const** {SearchTerm, lista}= **esta**.estado;

Pero ya que el libro utiliza JavaScript ES6 mayor parte del tiempo, usted debe atenerse a ella.

**Ceremonias:**

• leer más sobre [destructuring⁶⁵ ES6](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)

⁶⁵<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>

|  |  |
| --- | --- |
| Conceptos básicos en React | 60 |

**componentes controlados**

Ya has aprendido sobre el flujo de datos unidireccional en React. La misma ley se aplica para el campo de entrada, que actualiza el estado local con eltérmino de búsquedacon el fin de filtrar la lista. Cuando los cambios de estado, lahacer() método se ejecuta de nuevo y utiliza la reciente término de búsqueda desde el estado local para solicitar la condición de filtro.

Pero no olvidemos algo en el elemento de entrada? Una etiqueta HTML de entrada viene con unavaloratributo. El atributo de valor por lo general tiene el valor que se muestra en el campo de entrada. En este caso, sería eltérmino de búsquedapropiedad. Sin embargo, parece que no es necesario que en reaccionar.

Eso está mal. elementos de forma tal como<Input>, <Textarea> y <Select>mantener su propio estado en HTML plano. Modifican el valor internamente una vez que alguien cambia desde el exterior. En Reaccionar eso se llama un componente no controlada, ya que maneja su propio estado. En Reaccionar, usted debe asegurarse de hacer esos elementos componentes controlados.

¿Cómo se debe hacer eso? Es suficiente con establecer el atributo de valor del campo de entrada. El valor ya está guardado en eltérmino de búsquedapropiedad del Estado. ¿Por qué no acceder a ella a partir de ahí?

**src / app.js**

clase App extiende Componente {

...

render () {

const {searchTerm, lista} = this.state;

regreso (

<Div className = "App">

<Form>

<input

type = "text"

**value = {} searchTerm**

onChange = {} this.onSearchChange

/>

</ Form>

...

</ Div>

);

}

}

Eso es. El bucle de flujo de datos unidireccional para el campo de entrada es autónomo ahora. El estado del componente interno es la única fuente de la verdad para el campo de entrada.

|  |  |
| --- | --- |
| Conceptos básicos en React | 61 |

Toda la gestión de estado interno y el flujo unidireccional de datos podrían ser nuevas para usted. Pero una vez que están acostumbrados a ello, será su flujo natural para poner en práctica las cosas en React. En general, reaccionan traído un nuevo patrón con el flujo de datos unidireccional al mundo de las aplicaciones de una sola página. Se adoptó por varios marcos y bibliotecas por ahora.

**Ceremonias:**

• leer más sobre [reaccionar forms⁶⁶](https://facebook.github.io/react/docs/forms.html)

⁶⁶<https://facebook.github.io/react/docs/forms.html>

|  |  |
| --- | --- |
| Conceptos básicos en React | 62 |

**Componentes desglosadas**

Tiene un gran componente App ahora. Sigue creciendo y puede llegar a ser confuso con el tiempo.

Puede comenzar a dividirlo en trozos de componentes más pequeños.

Vamos a empezar a utilizar un componente para la entrada de búsqueda y un componente de la lista de elementos.

**src / app.js**

clase App extiende Componente {

...

render () {

const {searchTerm, lista} = this.state;

regreso (

<Div className = "App">

**<Búsqueda />**

**<Tabla />**

</ Div>

);

}

}

Puede pasar esas propiedades componentes que pueden utilizar ellos mismos. En el caso del componente de aplicación que necesita para pasar las propiedades administradas en el estado local y sus métodos de clase.

**src / app.js**

clase App extiende Componente {

...

render () {

const {searchTerm, lista} = this.state;

regreso (

<Div className = "App">

**<Buscar**

**value = {} searchTerm**

**onChange = {} this.onSearchChange**

**/>**

**<Tabla**

**list = {lista}**

|  |  |
| --- | --- |
| Conceptos básicos en React | 63 |

**patrón = {} searchTerm**

**onDismiss = {} this.onDismiss**

**/>**

</ Div>

);

}

}

Ahora se pueden definir los componentes del lado de su componente de aplicación. Estos componentes serán componentes de clase ES6 también. Ellos hacen que los mismos elementos como antes.

El primero es el componente de búsqueda.

**src / app.js**

clase App extiende Componente {

...

}

**Buscar extiende clase de componente {**

**render () {**

**const {valor, onChange} = this.props;**

**regreso (**

**<Form>**

**<input**

**type = "text"**

**value = {valor}**

**onChange = {} onChange**

**/>**

**</ Form>**

**);**

**}**

**}**

El segundo es el componente de la tabla.

|  |  |
| --- | --- |
| Conceptos básicos en React | 64 |

**src / app.js**

...

**Tabla de clase extiende Componente {**

**render () {**

**const {list, modelo, onDismiss} = this.props; regreso (**

**<Div>**

**{List.filter (isSearched (patrón)). Mapa (item =>**

**<Div clave = {} 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>**

**<botón**

**onclick = {() => onDismiss (item.objectID)}**

**type = "botón"**

**>**

**Despedir**

**</ Botón>**

**</ Span>**

**</ Div>**

**)}**

**</ Div>**

**);**

**}**

**}**

Ahora tiene tres componentes ES6 clase. Tal vez usted ha notado laaccesorios objeto que es accesible a través de la instancia de clase mediante el uso de esta. Los apoyos, forma abreviada de propiedades, tienen todos los valores que ha pasado a los componentes cuando se los utiliza en su componente de aplicación. De esta manera, los componentes pueden pasar propiedades abajo el árbol de componentes.

Mediante la extracción de los componentes del componente de aplicación, que sería capaz de volver a utilizarlos en otro lugar. Dado que los componentes obtienen sus valores mediante el uso de los puntales objeto, puede pasar cada vez que diferentes apoyos a sus componentes cuando se utilizan en otro lugar. Estos componentes se convirtieron reutilizable.

|  |  |
| --- | --- |
| Conceptos básicos en React | sesenta y cinco |

**Ceremonias:**

* averiguar otros componentes que se podía dividir como lo han hecho con los componentes de la búsqueda y la tabla

**-** pero no hacerlo ahora, de lo contrario se ejecutará en los conflictos en los próximos capítulos

|  |  |
| --- | --- |
| Conceptos básicos en React | 66 |

**componentes componibles**

Hay una más pequeña propiedad que se puede acceder en los apoyos objeto: la niñosapuntalar. Se puede utilizar para pasar a sus elementos componentes desde arriba, que son desconocidos para el propio componente, pero que sea posible componer los componentes entre sí. Vamos a ver cómo esto se ve como cuando pasa únicamente un texto (cadena) como un niño al componente de búsqueda.

**src / app.js**

clase App extiende Componente {

...

render () {

const {searchTerm, lista} = this.state;

regreso (

<Div className = "App">

**<Buscar**

**value = {} searchTerm**

**onChange = {} this.onSearchChange**

**>**

**Buscar**

**</ Buscar>**

<Tabla

list = {lista}

patrón = {} searchTerm

onDismiss = {} this.onDismiss

/>

</ Div>

);

}

}

Ahora el componente de búsqueda puede desestructurar la propiedad a los niños de los puntales objeto. Entonces se puede especificar dónde se deben mostrar a los niños.

|  |  |
| --- | --- |
| Conceptos básicos en React | 67 |

**src / app.js**

Buscar extiende clase de componente {

render () {

**const {valor, onChange, los niños} = this.props;**

regreso (

<Form>

**{} Niños <input**

type = "text"

value = {valor}

onChange = {} onChange

/>

</ Form>

);

}

}

El “Buscar” texto debe ser visible al lado de su campo de entrada ahora. Cuando se utiliza el componente de búsqueda en otro lugar, se puede elegir un texto diferente si lo desea. Después de todo, no es único texto que puede pasar cuando eran niños. Se puede pasar un elemento y árboles de elementos (que puede ser encapsulado por los componentes de nuevo) cuando eran niños. La propiedad niños hace que sea posible tejer componentes entre sí.

**Ceremonias:**

• leer más sobre [el modelo de composición de React⁶⁷](https://facebook.github.io/react/docs/composition-vs-inheritance.html)

⁶⁷<https://facebook.github.io/react/docs/composition-vs-inheritance.html>

|  |  |
| --- | --- |
| Conceptos básicos en React | 68 |

**Los componentes reutilizables**

Los componentes reutilizables y componibles le ayuden a subir con componentes capaces hier-archies. Son la base de la capa de vista de React. Los últimos capítulos mencionan el término reutilización. Puede volver a utilizar la tabla y los componentes de la búsqueda por ahora. Incluso el componente de aplicación es reutilizable, ya que podría crear una instancia que en algún otro lugar de nuevo.

Vamos a definir un componente más reutilizable, un componente Button, que consigue volver a utilizar con más frecuencia con el tiempo.

**src / app.js**

**clase** Botón **se extiende** Componente {

render () {

**const** {

al hacer clic,

nombre de la clase,

niños,

} = **esta**.accesorios;

**regreso** (

<botón

al hacer clic={al hacer clic}

nombre de la clase={nombre de la clase}

tipo="botón"

>

{niños}

</ Botón>

);

}

}

Podría parecer redundante para declarar un componente de este tipo. Que va a utilizar unaBotón componente en lugar de una botónelemento. Sólo se detiene eltype = "botón". Excepto por el tipo de atributo tiene que definir todo lo demás cuando se desea utilizar el componente Button. Pero hay que pensar en la inversión a largo plazo aquí. Imagine que tiene varios botones en su aplicación, pero quiere cambiar un atributo, el estilo o el comportamiento para el botón. Sin el componente que tendría que refactorizar cada botón. En cambio, el componente Button asegura tener sólo una única fuente de verdad. Un botón de refactorizar todos los botones a la vez. Un botón para gobernarlos a todos.

Puesto que usted ya tiene un elemento de botón, puede utilizar el componente Button en lugar. Se omite el atributo de tipo, debido a que el componente Button lo especifica.

|  |  |
| --- | --- |
| Conceptos básicos en React | 69 |

**src / app.js**

Tabla de clase extiende Componente {

render () {

const {list, modelo, onDismiss} = this.props; regreso (

<Div>

{List.filter (isSearched (patrón)). Mapa (item =>

<Div clave = {} 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>

**<Botón onclick = {() => onDismiss (item.objectID)}> Descartar**

**</ Botón>**

</ Span>

</ Div>

)}

</ Div>

);

}

}

El componente Button espera una nombre de la clasepropiedad en los apoyos. losnombre de la claseatributo es otra Reaccionar derivado de la clase de atributo HTML. Pero no pasó ningunanombre de la clasecuando se utilizó el botón. En el código que debería ser más explícito en el componente Button que lanombre de la clase es opcional.

Por lo tanto, se puede utilizar el parámetro por defecto que es una función de JavaScript ES6.

|  |  |
| --- | --- |
| Conceptos básicos en React | 70 |

**src / app.js**

Botón clase extiende Componente {

render () {

{const

al hacer clic,

**className = '',**

niños,

} = This.props;

...

}

}

Ahora, siempre que no haya nombre de la clase propiedad especificada cuando se utiliza el componente Button, el valor será una cadena vacía en lugar de indefinido.

**Ceremonias:**

• leer más sobre [ES6 predeterminado parameters⁶⁸](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Default_parameters)

⁶⁸<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Default_parameters>

|  |  |
| --- | --- |
| Conceptos básicos en React | 71 |

**Declaraciones de componentes**

A estas alturas ya tiene cuatro componentes de la clase ES6. Pero se puede hacer mejor. Déjeme introducir componentes funcionales sin estado como alternativa para los componentes de la clase ES6. Antes de que se refactorizar sus componentes, vamos a introducir los diferentes tipos de componentes en React.

* **Componentes funcionales sin estado:** Estos componentes son funciones que obtienen una entrada y una salida devuelven. La entrada son los puntales. La salida es una instancia de componente así llanura JSX. Hasta ahora, es bastante similar a un componente de clase ES6. Sin embargo, los componentes sin estado funcional son funciones (funcional) y no tienen ningún estado local (stateless). No se puede acceder o actualizar el estado coneste estado o this.setState () porque no hay estaobjeto. Además, no tienen métodos de ciclo de vida. No aprendiste acerca de los métodos de ciclo de vida todavía, pero ya ha utilizado dos:constructor() y hacer(). Mientras que el constructor se ejecuta sólo una vez en la vida útil de un componente, elhacer()método de clase se ejecuta una vez en el principio y cada vez que las actualizaciones de componentes. Tenga en cuenta que los componentes funcionales sin estado no tienen métodos de ciclo de vida, cuando esté en el capítulo métodos de ciclo de vida más adelante.
* **Componentes de la Clase ES6:** Ya ha utilizado este tipo de declaración de componentes en sus cuatro componentes. En la definición de clase, que se extienden desde el componente React. losampliarganchos todos los métodos de ciclo de vida, disponibles en la API de componente reaccionar, al componente. De esa manera usted fuera capaz de utilizar elhacer()método de clase. Además, puede almacenar y manipular el estado de los componentes de clase mediante el uso de ES6este estado y this.setState ().
* **React.createClass:** La declaración de componentes se utilizó en versiones anteriores de reaccionar y todavía en JavaScript ES5 Reaccionar aplicaciones. Pero[Facebook lo declaró como deprecated⁶⁹](https://facebook.github.io/react/blog/2015/03/10/react-v0.13.html) a favor de JavaScript ES6. Incluso agregaron una[advertencia desaprobación en la versión 15.5⁷⁰](https://facebook.github.io/react/blog/2017/04/07/react-v15.5.0.html). Usted no va a usar en el libro.

Así que, básicamente, sólo hay dos declaraciones de componentes que quedan. Pero cuándo utilizar componentes funcionales sin estado más de los componentes de clase ES6? Una regla de oro es usar componentes funcionales sin estado cuando no se necesita métodos estatales o componente del ciclo de vida locales. Por lo general, se empieza a poner en práctica sus componentes como componentes funcionales sin estado. Una vez que necesita tener acceso a los métodos del estado o del ciclo de vida, lo que tiene que refactorizar a un componente de clase ES6. En nuestra aplicación, empezamos al revés por el bien de aprendizaje reaccionar.

Volvamos a la aplicación. El componente de aplicación utiliza estado interno. Es por eso que tiene que permanecer como un componente de clase ES6. Pero los otros tres de sus componentes son de clase ES6 sin estado. Ellos no necesitan tener acceso aeste estado o this.setState (). Aún más, no tienen métodos de ciclo de vida. Vamos a refactorizar juntos el componente de búsqueda de un componente funcional sin estado. La Mesa y botón del componente refactorización permanecerá como su ejercicio.

⁶⁹<https://facebook.github.io/react/blog/2015/03/10/react-v0.13.html>

⁷⁰<https://facebook.github.io/react/blog/2017/04/07/react-v15.5.0.html>

|  |  |
| --- | --- |
| Conceptos básicos en React | 72 |

**src / app.js**

**Función de búsqueda (apoyos) {**

**const {valor, onChange, los niños} = puntales; regreso (**

**<Form>**

**{} Niños <input**

**type = "text"**

**value = {valor}**

**onChange = {} onChange**

**/>**

**</ Form>**

**);**

**}**

Eso es básicamente la misma. Los apoyos son accesibles en la firma de la función y el valor de retorno es JSX. Pero se puede hacer más sabia código en un componente sin estado funcional. Usted ya conoce la desestructuración ES6. Lo más recomendable es usarlo en la firma de la función a desestructurar los puntales.

**src / app.js**

**Función de búsqueda ({valor, onChange, los niños}) {**

regreso (

<Form>

{} Niños <input

type = "text"

value = {valor}

onChange = {} onChange

/>

</ Form>

);

}

Pero puede mejorar. Usted ya sabe que las funciones de dirección ES6 le permiten mantener sus funciones concisa. Se puede quitar el cuerpo del bloque de la función. En un cuerpo concisa rendimiento, implícito se une de este modo se puede quitar la instrucción de retorno. Desde su componente sin estado funcional es una función, se puede mantener concisa también.

|  |  |
| --- | --- |
| Conceptos básicos en React | 73 |

**src / app.js**

**Buscar const = ({valor, onChange, los niños}) => <form>**

**{} Niños <input**

**type = "text"**

**value = {valor}**

**onChange = {} onChange**

**/>**

**</ Form>**

El último paso fue especialmente útil para hacer cumplir sólo para tener apoyos como entrada y JSX como salida. Nada en el medio. Aún así, se puede hacer algo en el medio mediante el uso de un cuerpo de bloque en su función de flecha ES6.

**código de Área de juegos**

**const** Buscar=({Valor, onChange, los niños}) => {

*// hacer algo*

**regreso** (

<formar>

{niños} <entrada

tipo="texto"

valor={valor}

onChange={} OnChange

/>

</ Form>

);

}

Pero usted no lo necesita por ahora. Es por eso que se puede mantener la versión anterior sin el cuerpo del bloque. Al utilizar cuerpos de bloque, la gente a menudo tienden a hacer demasiadas cosas en la función. Al dejar el cuerpo del bloque hacia fuera, usted puede centrarse en la entrada y salida de su función.

Ahora usted tiene un componente sin estado funcional de peso ligero. Una vez que se necesita el acceso a sus métodos de ciclo de vida del estado componente o internos, que le refactorearlo a un componente de clase ES6. Además hemos visto cómo JavaScript ES6 se puede utilizar en React componentes para hacerlos más concisa y elegante.

|  |  |
| --- | --- |
| Conceptos básicos en React | 74 |

**Ceremonias:**

* refactorizar el componente Tabla y Button a los apátridas componentes funcionales
* leer más sobre [componentes de la clase ES6 y components⁷¹ sin estado funcional](https://facebook.github.io/react/docs/components-and-props.html)

⁷¹<https://facebook.github.io/react/docs/components-and-props.html>

|  |  |
| --- | --- |
| Conceptos básicos en React | 75 |

**componentes de estilo**

Let’s add some basic styling to your application and components. You can reuse the src/App.css and src/index.css files. These files should already be in your project since you have bootstrapped it with create-react-app. They should be imported in your src/App.js and src/index.js files too. I prepared some CSS which you can simply copy and paste to these files, but feel free to use your own style at this point.

First, styling for your overall application.

**src/index.css**

**body** {

**color**:#222;

**background**:#f4f4f4;

**font**:400 14**px** CoreSans, Arial,**sans-serif**;

}

* {

**color**:#222;

}

**a**:hover{

**text-decoration**: **underline**;

}

**ul**, **li** {

**list-style**: **none**;

**padding**:0;

**margin**:0;

}

**input** {

**padding**:10**px**;

**border-radius**:5**px**;

**outline**: **none**;

**margin-right**:10**px**;

**border**:1**px solid** #dddddd;

}

**button** {

**padding**:10**px**;

**border-radius**:5**px**;

|  |  |
| --- | --- |
| Basics in React | 76 |

**border**:1**px solid** #dddddd;

**background**: **transparent**;

**color**:#808080;

**cursor**: **pointer**;

}

**button**:hover{

**color**:#222;

}

\*:focus {

**outline**: **none**;

}

Second, styling for your components in the App file.

**src/App.css**

.**page** {

**margin**:20**px**;

}

.**interactions** {

**text-align**: **center**;

}

.**table** {

**margin**:20**px** 0;

}

.**table-header** {

**display**: **flex**;

**line-height**:24**px**;

**font-size**:16**px**;

**padding**:0 10**px**;

**justify-content**: **space-between**;

}

.**table-empty** {

**margin**:200**px**;

**text-align**: **center**;

**font-size**:16**px**;

}

|  |  |
| --- | --- |
| Basics in React | 77 |

.**table-row** {

**display**: **flex**;

**line-height**:24**px**;

**white-space**: **nowrap**;

**margin**:10**px** 0;

**padding**:10**px**;

**background**:#ffffff;

**border**:1**px solid** #e3e3e3;

}

.**table-header** > **span** {

**overflow**: **hidden**;

**text-overflow**: **ellipsis**;

**padding**:0 5**px**;

}

.**table-row** > **span** {

**overflow**: **hidden**;

**text-overflow**: **ellipsis**;

**padding**:0 5**px**;

}

.**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**:1**px solid** #38BB6C;

}

Now you can use the style in some of your components. Don’t forget to use React className instead of class as HTML attribute.

First, apply it in your App ES6 class component.

|  |  |
| --- | --- |
| Basics in React | 78 |

**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>**

);

}

}

Second, apply it in your Table functional stateless component.

**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>**

Now you have styled your application and components with basic CSS. It should look quite decent. As you know, JSX mixes up HTML and JavaScript. Now one could argue to add CSS in the mix as well. That’s called inline style. You can define JavaScript objects and pass them to the style attribute of an element.

Let’s keep the Table column width flexible by using inline style.

**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>

The style is inlined now. You could define the style objects outside of your elements to make it cleaner.

**Code Playground**

**const** largeColumn={

width: '40%',

};

**const** midColumn={

width: '30%',

};

**const** smallColumn={

width: '10%',

};

After that you would use them in your columns: <span style={smallColumn}>.

In general, you will find different opinions and solutions for style in React. You used pure CSS and inline style now. That’s sufficient to get started.

I don’t want to be opinionated here, but I want to leave you some more options. You can read about them and apply them on your own. But if you are new to React, I would recommend to stick to pure CSS and inline style for now.

* [styled-components⁷²](https://github.com/styled-components/styled-components)
* [CSS Modules⁷³](https://github.com/css-modules/css-modules)

⁷²<https://github.com/styled-components/styled-components>

⁷³<https://github.com/css-modules/css-modules>

|  |  |
| --- | --- |
| Basics in React | 81 |

You have learned the basics to write your own React application! Let’s recap the last chapters:

* React

**–** use this.stateand setState()to manage your internal component state

**–** pass functions or class methods to your element handler

**–** use forms and events in React to add interactions

**–** unidirectional data flow is an important concept in React

**–** embrace controlled components

**–** compose components with children and reusable components

**–** usage and implementation of ES6 class components and functional stateless components

**–** approaches to style your components

* ES6

**–** functions that are bound to a class are class methods

**–** destructuring of objects and arrays

**–** default parameters

* General

**–** higher-order functions

Again it makes sense to take a break. Internalize the learnings and apply them on your own. You can experiment with the source code you have written so far. Additionally you can read more in the official [documentation⁷⁴](https://facebook.github.io/react/docs/installation.html).

You can find the source code in the [official repository⁷⁵](https://github.com/rwieruch/hackernews-client/tree/4.2).

⁷⁴<https://facebook.github.io/react/docs/installation.html>

⁷⁵<https://github.com/rwieruch/hackernews-client/tree/4.2>

**Getting Real with an API**

Now it’s time to get real with an API, because it can get boring to deal with sample data.

If you are not familiar with APIs, I encourage you [to read my journey where I got to know APIs⁷⁶](https://www.robinwieruch.de/what-is-an-api-javascript/).

Do you know the [Hacker News⁷⁷](https://news.ycombinator.com/) platform? It’s a great news aggregator about tech topics. In this book, you will use the Hacker News API to fetch trending stories from the platform. There is a [basic⁷⁸](https://github.com/HackerNews/API) and [search⁷⁹](https://hn.algolia.com/api) API to get data from the platform. The latter one makes sense in the case of this application in order to search stories on Hacker News. You can visit the API specification to get an understanding of the data structure.

⁷⁶<https://www.robinwieruch.de/what-is-an-api-javascript/>

⁷⁷<https://news.ycombinator.com/>

⁷⁸<https://github.com/HackerNews/API>

⁷⁹<https://hn.algolia.com/api>

|  |  |
| --- | --- |
| Getting Real with an API | 83 |

**Lifecycle Methods**

You will need to know about React lifecycle methods before you can start to fetch data in your components by using an API. These methods are a hook into the lifecycle of a React component. They can be used in ES6 class components, but not in functional stateless components.

Do you remember when a previous chapter taught you about JavaScript ES6 classes and how they are used in React? Apart from the render() method, there are several methods that can be overridden in a React ES6 class component. All of these are the lifecycle methods. Let’s dive into them:

You already know two lifecycle methods that can be used in an ES6 class component: constructor() and render().

The constructor is only called when an instance of the component is created and inserted in the DOM. The component gets instantiated. That process is called mounting of the component.

The render() method is called during the mount process too, but also when the component updates. Each time when the state or the props of a component change, the render() method of the component is called.

Now you know more about the two lifecycle methods and when they are called. You already used them as well. But there are more of them.

The mounting of a component has two more lifecycle methods: componentWillMount() and componentDidMount(). The constructor is called first, componentWillMount() gets called before the

render() method and componentDidMount() is called after the render() method.

Overall the mounting process has 4 lifecycle methods. They are invoked in the following order:

* constructor()
* componentWillMount()
* render()
* componentDidMount()

But what about the update lifecycle of a component that happens when the state or the props change?

Overall it has 5 lifecycle methods in the following order:

* componentWillReceiveProps()
* shouldComponentUpdate()
* componentWillUpdate()
* render()
* componentDidUpdate()

|  |  |
| --- | --- |
| Getting Real with an API | 84 |

Last but not least there is the unmounting lifecycle. It has only one lifecycle method: componen-tWillUnmount().

After all, you don’t need to know all of these lifecycle methods from the beginning. It can be intimidating yet you will not use all of them. Even in a larger React application you will only use a few of them apart from the constructor() and the render() method. Still, it is good to know that each lifecycle method can be used for specific use cases:

* **constructor(props)** - It is called when the component gets initialized. You can set an initial component state and bind class methods during that lifecycle method.
* **componentWillMount()** - It is called before the render()lifecycle method. That’s why it could be used to set internal component state, because it will not trigger a second rendering of the component. Generally it is recommended to use the constructor() to set the initial state.
* **render()** - This lifecycle method is mandatory and returns the elements as an output of the component. The method should be pure and therefore shouldn’t modify the component state. It gets an input as props and state and returns an element.
* **componentDidMount()** - It is called only once when the component mounted. That’s the perfect time to do an asynchronous request to fetch data from an API. The fetched data would get stored in the internal component state to display it in the render() lifecycle method.
* **componentWillReceiveProps(nextProps)** - The lifecycle method is called during an update lifecycle. As input you get the next props. You can diff the next props with the previous props, by using this.props, to apply a different behavior based on the diff. Additionally, you can set state based on the next props.
* **shouldComponentUpdate(nextProps, nextState)** - It is always called when the component updates due to state or props changes. You will use it in mature React applications for performance optimizations. Depending on a boolean that you return from this lifecycle method, the component and all its children will render or will not render on an update lifecycle. You can prevent the render lifecycle method of a component.
* **componentWillUpdate(nextProps, nextState)** - The lifecycle method is immediately in-voked before the render() method. You already have the next props and next state at your disposal. You can use the method as last opportunity to perform preparations before the render method gets executed. Note that you cannot trigger setState() anymore. If you want to compute state based on the next props, you have to use componentWillReceiveProps().
* **componentDidUpdate(prevProps, prevState)** - The lifecycle method is immediately invoked after the render() method. You can use it as opportunity to perform DOM operations or to perform further asynchronous requests.
* **componentWillUnmount()** - It is called before you destroy your component. You can use the lifecycle method to perform any clean up tasks.

The constructor() and render() lifecycle methods are already used by you. These are the commonly used lifecycle methods for ES6 class components. Actually the render() method is required, otherwise you wouldn’t return a component instance.

|  |  |
| --- | --- |
| Getting Real with an API | 85 |

There is one more lifecycle method: componentDidCatch(error, info). It was introduced in [React](https://www.robinwieruch.de/what-is-new-in-react-16/) [16⁸⁰](https://www.robinwieruch.de/what-is-new-in-react-16/) and is used to catch errors in components. For instance, displaying the sample list in your application works just fine. But there could be a case when the list in the local state is set to null by accident (e.g. when fetching the list from an external API, but the request failed and you set the local state of the list to null). Afterward, it wouldn’t be possible to filter and map the list anymore, because it is null and not an empty list. The component would be broken and the whole application would fail. Now, by using componentDidCatch(), you can catch the error, store it in your local state, and show an optional message to your application user that an error has happened.

**Exercises:**

* read more about [lifecycle methods in React⁸¹](https://facebook.github.io/react/docs/react-component.html)
* read more about [the state related to lifecycle methods in React⁸²](https://facebook.github.io/react/docs/state-and-lifecycle.html)
* read more about [error handling in components⁸³](https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html)

⁸⁰<https://www.robinwieruch.de/what-is-new-in-react-16/>

⁸¹<https://facebook.github.io/react/docs/react-component.html>

⁸²<https://facebook.github.io/react/docs/state-and-lifecycle.html>

⁸³<https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html>

|  |  |
| --- | --- |
| Getting Real with an API | 86 |

**Fetching Data**

Now you are prepared to fetch data from the Hacker News API. There was one lifecycle method mentioned that can be used to fetch data: componentDidMount(). You will use the native fetch API in JavaScript to perform the request.

Before we can use it, let’s set up the URL constants and default parameters to breakup the API request into chunks.

**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=';**

...

In JavaScript ES6, you can use [template strings⁸⁴](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Template_literals) to concatenate strings. You will use it to concatenate your URL for the API endpoint.

**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*

That will keep your URL composition flexible in the future.

But let’s get to the API request where you will use the url. The whole data fetch process will be presented at once, but each step will be explained afterward.

⁸⁴<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Template_literals>

|  |  |
| --- | --- |
| Getting Real with an API | 87 |

**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.onDismiss = this.onDismiss.bind(this);

}

**setSearchTopStories(result) {**

**this.setState({ result });**

**}**

**fetchSearchTopStories(searchTerm) {**

**fetch(`${PATH\_BASE}${PATH\_SEARCH}?${PARAM\_SEARCH}${searchTerm}`)**

**.then(response => response.json())**

**.then(result => this.setSearchTopStories(result))**

**.catch(e => e);**

**}**

**componentDidMount() {**

**const { searchTerm } = this.state;**

**this.fetchSearchTopStories(searchTerm);**

**}**

...

}

A lot of things happen in the code. I thought about breaking it into smaller pieces. Then again it would be difficult to grasp the relations of each piece to each other. Let me explain each step in detail.

|  |  |
| --- | --- |
| Getting Real with an API | 88 |

First, you can remove the sample list of items, because you return a real list from the Hacker News API. The sample data is not used anymore. The initial state of your component has an empty result and default search term now. The same default search term is used in the input field of the Search component and in your first request.

Second, you use the componentDidMount() lifecycle method to fetch the data after the component did mount. In the very first fetch, the default search term from the local state is used. It will fetch “redux” related stories, because that is the default parameter.

Third, the native fetch API is used. The JavaScript ES6 template strings allow it to compose the URL with the searchTerm. The URL is the argument for the native fetch API function. The response needs to get transformed to a JSON data structure, which is a mandatory step in a native fetch function when dealing with JSON data structures, and can finally be set as result in the internal component state. In addition, the catch block is used in case of an error. If an error happens during the request, the function will run into the catch block instead of the then block. In a later chapter of the book, you will include the error handling.

Last but not least, don’t forget to bind your new component methods in the constructor.

Now you can use the fetched data instead of the sample list of items. However, you have to be careful again. The result is not only a list of data. [It’s a complex object with meta information and a list of hits](https://hn.algolia.com/api) [which are in our case the stories⁸⁵](https://hn.algolia.com/api). You can output the internal state with console.log(this.state); in your render() method to visualize it.

In the next step, you will use the result to render it. But we will prevent it from rendering anything, so we will return null, when there is no result in the first place. Once the request to the API succeeded, the result is saved to the state and the App component will re-render with the updated state.

**src/App.js**

class App extends Component {

...

render() {

**const { searchTerm, result } = this.state;**

**if (!result) { return null; }**

return (

<div className="page">

...

<Table

**list={result.hits}**

pattern={searchTerm}

⁸⁵<https://hn.algolia.com/api>

|  |  |
| --- | --- |
| Getting Real with an API | 89 |

onDismiss={this.onDismiss}

/>

</div>

);

}

}

Let’s recap what happens during the component lifecycle. Your component gets initialized by the constructor. After that, it renders for the first time. But you prevent it from displaying anything, because the result in the local state is null. It is allowed to return null for a component in order to display nothing. Then the componentDidMount() lifecycle method runs. In that method you fetch the data from the Hacker News API asynchronously. Once the data arrives, it changes your internal component state in setSearchTopStories(). Afterward, the update lifecycle comes into play because the local state was updated. The component runs the render() method again, but this time with populated result in your internal component state. The component and thus the Table component with its content will be rendered.

You used the native fetch API that is supported by most browsers to perform an asynchronous request to an API. The create-react-app configuration makes sure that it is supported in every browser. There are third-party node packages that you can use to substitute the native fetch API: [superagent⁸⁶](https://github.com/visionmedia/superagent) and [axios⁸⁷](https://github.com/mzabriskie/axios).

Keep in mind that the book builds up on the JavaScript’s shorthand notation for truthfulness checks. In the previous example, if (!result) was used in favor of if (result === null). The same applies for other cases throughout the book too. For instance, if (!list.length) is used in favor of if (list.length === 0) or if (someString) is used in favor of if (someString !== ''). Read up about the topic if you are not too familiar with it.

Back to your application: The list of hits should be visible now. However, there are two regression bugs in the application now. First, the “Dismiss” button is broken. It doesn’t know about the complex result object and still operates on the plain list from the local state when dismissing an item. Second, when the list is displayed but you try to search for something else, the list gets filtered on the client-side even though the initial search was made by searching for stories on the server-side. The perfect behvaior would be to fetch another result object from the API when using the Search component. Both regression bugs will be fixed in the following chapters.

**Exercises:**

* read more about [ES6 template strings⁸⁸](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Template_literals)
* read more about [the native fetch API⁸⁹](https://developer.mozilla.org/en/docs/Web/API/Fetch_API)

⁸⁶<https://github.com/visionmedia/superagent>

⁸⁷<https://github.com/mzabriskie/axios>

⁸⁸<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Template_literals>

⁸⁹<https://developer.mozilla.org/en/docs/Web/API/Fetch_API>

|  |  |
| --- | --- |
| Getting Real with an API | 90 |

• read more about [data fetching in React⁹⁰](https://www.robinwieruch.de/react-fetching-data/)

⁹⁰<https://www.robinwieruch.de/react-fetching-data/>

|  |  |
| --- | --- |
| Getting Real with an API | 91 |

**ES6 Spread Operators**

The “Dismiss” button doesn’t work because the onDismiss() method is not aware of the complex result object. It only knows about a plain list in the local state. But it isn’t a plain list anymore. Let’s change it to operate on the result object instead of the list itself.

**src/App.js**

onDismiss(id) {

const isNotId = item => item.objectID !== id;

**const updatedHits = this.state.result.hits.filter(isNotId); this.setState({**

**...**

**});**

}

But what happens in setState() now? Unfortunately the result is a complex object. The list of hits is only one of multiple properties in the object. However, only the list gets updated, when an item gets removed in the result object, while the other properties stay the same.

One approach could be to mutate the hits in the result object. I will demonstrate it, but we won’t do it that way.

**Code Playground**

*// don`t do this*

**this**.state.result.hits=updatedHits;

React embraces immutable data structures. Thus you shouldn’t mutate an object (or mutate the state directly). A better approach is to generate a new object based on the information you have. Thereby none of the objects get altered. You will keep the immutable data structures. You will always return a new object and never alter an object.

Therefore you can use JavaScript ES6 Object.assign(). It takes as first argument a target object. All following arguments are source objects. These objects are merged into the target object. The target object can be an empty object. It embraces immutability, because no source object gets mutated. It would look similar to the following:

**Code Playground**

**const** updatedHits={ hits:updatedHits };

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

Latter objects will override former merged objects when they share the same property names. Now let’s do it in the onDismiss() method:

|  |  |
| --- | --- |
| Getting Real with an API | 92 |

**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 })** });

}

That would already be the solution. But there is a simpler way in JavaScript ES6 and future JavaScript releases. May I introduce the spread operator to you? It only consists of three dots: ... When it is used, every value from an array or object gets copied to another array or object.

Let’s examine the ES6 array spread operator even though you don’t need it yet.

**Code Playground**

**const** userList=['Robin','Andrew','Dan']; **const** additionalUser='Jordan';

**const** allUsers=[ ...userList, additionalUser ];

console.log(allUsers);

*// output: ['Robin', 'Andrew', 'Dan', 'Jordan']*

The allUsers variable is a completely new array. The other variables userList and additionalUser stay the same. You can even merge two arrays that way into a new array.

**Code Playground**

**const** oldUsers=['Robin','Andrew'];

**const** newUsers=['Dan','Jordan'];

**const** allUsers=[ ...oldUsers, ...newUsers ];

console.log(allUsers);

*// output: ['Robin', 'Andrew', 'Dan', 'Jordan']*

Now let’s have a look at the object spread operator. It is not JavaScript ES6. It is a [proposal for a](https://github.com/sebmarkbage/ecmascript-rest-spread) [next JavaScript version⁹¹](https://github.com/sebmarkbage/ecmascript-rest-spread) yet already used by the React community. That’s why *create-react-app* incorporated the feature in the configuration.

Basically it is the same as the JavaScript ES6 array spread operator but with objects. It copies each key value pair into a new object.

⁹¹<https://github.com/sebmarkbage/ecmascript-rest-spread>

|  |  |
| --- | --- |
| Getting Real with an API | 93 |

**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 }*

Multiple objects can be spread like in the array spread example.

**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 }*

After all, it can be used to replace Object.assign().

**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 }** });

}

Now the “Dismiss” button should work again, because the onDismiss() method is aware of the complex result object and how to update it after dismissing an item from the list.

**Exercises:**

* read more about the [ES6 Object.assign()⁹²](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/assign)
* read more about the [ES6 array spread operator⁹³](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Spread_operator)

**–** the object spread operator is briefly mentioned

⁹²<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/assign> ⁹³<https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Spread_operator>

|  |  |
| --- | --- |
| Getting Real with an API | 94 |

**Conditional Rendering**

Conditional rendering is introduced pretty early in React applications. But not in the case of the book, because there wasn’t such an use case yet. The conditional rendering happens when you want to make a decision to render either one or another element. Sometimes it means to render an element or nothing. After all, a conditional rendering simplest usage can be expressed by an if-else statement in JSX.

The result object in the internal component state is null in the beginning. So far, the App component returned no elements when the result hasn’t arrived from the API. That’s already a conditional rendering, because you return earlier from the render() lifecycle method for a certain condition. The App component either renders nothing or its elements.

But let’s go one step further. It makes more sense to wrap the Table component, which is the only component that depends on the result, in an independent conditional rendering. Everything else should be displayed, even though there is no result yet. You can simply use a ternary operator in your JSX.

**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 | 95 |

</div>

);

}

}

That’s your second option to express a conditional rendering. A third option is the logical && operator.

In JavaScript a true && 'Hello World' always evaluates to ‘Hello World’. A false && 'Hello

World' always evaluates to false.

**Code Playground**

**const** result= **true** &&'Hello World';

console.log(result);

*// output: Hello World*

**const** result= **false** &&'Hello World';

console.log(result);

*// output: false*

In React you can make use of that behavior. If the condition is true, the expression after the logical

* operator will be the output. If the condition is false, React ignores and skips the expression. It is applicable in the Table conditional rendering case, because it should return a Table or nothing.

**src/App.js**

{ result && <Table

list={result.hits} pattern={searchTerm} onDismiss={**this**.onDismiss}

/>

}

These were a few approaches to use conditional rendering in React. You can read about [more](https://www.robinwieruch.de/conditional-rendering-react/) [alternatives in an exhaustive list of examples for conditional rendering approaches⁹⁴](https://www.robinwieruch.de/conditional-rendering-react/). Moreover you will get to know their different use cases and when to apply them.

After all, you should be able to see the fetched data in your application. Everything except the Table is displayed when the data fetching is pending. Once the request resolves the result and stores it into the local state, the Table is displayed because the render() method runs again and the condition in the conditional rendering resolves in favor of displaying the Table component.

⁹⁴<https://www.robinwieruch.de/conditional-rendering-react/>

|  |  |
| --- | --- |
| Getting Real with an API | 96 |

**Exercises:**

* read more about [React conditional rendering⁹⁵](https://facebook.github.io/react/docs/conditional-rendering.html)
* read more about [different ways for conditional renderings⁹⁶](https://www.robinwieruch.de/conditional-rendering-react/)

⁹⁵<https://facebook.github.io/react/docs/conditional-rendering.html>

⁹⁶<https://www.robinwieruch.de/conditional-rendering-react/>

|  |  |
| --- | --- |
| Getting Real with an API | 97 |

**Client- or Server-side Search**

When you use the Search component with its input field now, you will filter the list. That’s happening on the client-side though. Now you are going to use the Hacker News API to search on the server-side. Otherwise you would deal only with the first API response which you got on componentDidMount() with the default search term parameter.

You can define an onSearchSubmit() method in your App component which fetches results from the Hacker News API when executing a search in the Search component. It will be the same fetch as in your componentDidMount() lifecycle method, but this time with a modified search term from the local state and not with the initial default search term.

**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);

}

...

**onSearchSubmit() {**

**const { searchTerm } = this.state;**

**this.fetchSearchTopStories(searchTerm);**

**}**

...

}

Now the Search component has to add an additional button. The button has to explicitly trigger the search request. Otherwise you would fetch data from the Hacker News API every time when your input field changes. But you want to do it explicitly in a onClick() handler.

|  |  |
| --- | --- |
| Getting Real with an API | 98 |

As alternative you could debounce (delay) the onChange() function and spare the button, but it would add more complexity at this time and maybe wouldn’t be the desired effect. Let’s keep it simple without a debounce for now.

First, pass the onSearchSubmit() method to your Search component.

**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>

);

}

}

Second, introduce a button in your Search component. The button has the type="submit" and the form uses its onSubmit() attribute to pass the onSubmit() method. You can reuse the children property, but this time it will be used as the content of the button.

|  |  |
| --- | --- |
| Getting Real with an API | 99 |

**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>**

In the Table, you can remove the filter functionality, because there will be no client-side filter (search) anymore. Don’t forget to remove the isSearched() function as well. It will not be used anymore. The result comes directly from the Hacker News API now after you have clicked the “Search” button.

**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>

When you try to search now, you will notice that the browser reloads. That’s a native browser behavior for a submit callback in a HTML form. In React you will often come across the preventDefault() event method to suppress the native browser behavior.

**src/App.js**

**onSearchSubmit(event) {**

const { searchTerm } = this.state;

this.fetchSearchTopStories(searchTerm);

**event.preventDefault();**

}

Now you should be able to search different Hacker News stories. Perfect, you interact with a real world API. There should be no client-side search anymore.

**Exercises:**

* read more about [synthetic events in React⁹⁷](https://facebook.github.io/react/docs/events.html)
* experiment with the [Hacker News API⁹⁸](https://hn.algolia.com/api)

⁹⁷<https://facebook.github.io/react/docs/events.html>

⁹⁸<https://hn.algolia.com/api>

|  |  |
| --- | --- |
| Getting Real with an API | 101 |

**Paginated Fetch**

Did you have a closer look at the returned data structure yet? The [Hacker News API⁹⁹](https://hn.algolia.com/api) returns more than a list of hits. Precisely it returns a paginated list. The page property, which is 0 in the first response, can be used to fetch more paginated sublists as result. You only need to pass the next page with the same search term to the API.

Let’s extend the composable API constants so that it can deal with paginated data.

**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=';**

Now you can use the new constant to add the page parameter to your API request.

**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=*

The fetchSearchTopStories() method will take the page as second argument. If you don’t provide the second argument, it will fallback to the 0 page for the initial request. Thus the componentDidMount() and onSearchSubmit() methods fetch the first page on the first request. Every additional fetch should fetch the next page by providing the second argument.

⁹⁹<https://hn.algolia.com/api>

|  |  |
| --- | --- |
| Getting Real with an API | 102 |

**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(e => e);

}

...

}

Now you can use the current page from the API response in fetchSearchTopStories(). You can use this method in a button to fetch more stories on a onClick button handler. Let’s use the Button to fetch more paginated data from the Hacker News API. You only need to define the onClick() handler which takes the current search term and the next page (current page + 1).

**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">**

|  |  |
| --- | --- |
| Getting Real with an API | 103 |

**<Button onClick={() => this.fetchSearchTopStories(searchTerm, page + 1\**

**)}>**

**More**

**</Button>**

**</div>**

</div>

);

}

}

In addition, in your render() method you should make sure to default to page 0 when there is no result yet. Remember that the render() method is called before the data is fetched asynchronously in the componentDidMount() lifecycle method.

There is one step missing. You fetch the next page of data, but it will override your previous page of data. It would be ideal to concatenate the old and new list of hits from the local state and new result object. Let’s adjust the functionality to add the new data rather than to override it.

**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 }**

**});**

}

A couple of things happen in the setSearchTopStories() method now. First, you get the hits and page from the result.

Second, you have to check if there are already old hits. When the page is 0, it is a new search request from componentDidMount() or onSearchSubmit(). The hits are empty. But when you click the “More” button to fetch paginated data the page isn’t 0. It is the next page. The old hits are already stored in your state and thus can be used.

|  |  |
| --- | --- |
| Getting Real with an API | 104 |

Third, you don’t want to override the old hits. You can merge old and new hits from the recent API request. The merge of both lists can be done with the JavaScript ES6 array spread operator.

Fourth, you set the merged hits and page in the local component state.

You can make one last adjustment. When you try the “More” button it only fetches a few list items. The API URL can be extended to fetch more list items with each request. Again, you can add more composable path constants.

**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=';**

Now you can use the constants to extend the API URL.

**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(e => e);

}

Afterward, the request to the Hacker News API fetches more list items in one request than before. As you can see, a powerful API such as the Hacker News API gives you plenty of ways to experiment with real world data. You should make use of it to make your endeavours when learning something new more exciting. That’s [how I learned about the empowerment that APIs provide¹⁰⁰](https://www.robinwieruch.de/what-is-an-api-javascript/) when learning a new programming language or library.

**Exercises:**

• experiment with the [Hacker News API parameters¹⁰¹](https://hn.algolia.com/api)

¹⁰⁰<https://www.robinwieruch.de/what-is-an-api-javascript/>

¹⁰¹<https://hn.algolia.com/api>

|  |  |
| --- | --- |
| Getting Real with an API | 105 |

**Client Cache**

Each search submit makes a request to the Hacker News API. You might search for “redux”, followed by “react” and eventually “redux” again. In total it makes 3 requests. But you searched for “redux” twice and both times it took a whole asynchronous roundtrip to fetch the data. In a client-sided cache you would store each result. When a request to the API is made, it checks if a result is already there. If it is there, the cache is used. Otherwise an API request is made to fetch the data.

In order to have a client cache for each result, you have to store multiple results rather than one result in your internal component state. The results object will be a map with the search term as key and the result as value. Each result from the API will be saved by search term (key).

At the moment, your result in the local state looks similar to the following:

**Code Playground**

result: {

hits: [ ... ],

page: 2,

}

Imagine you have made two API requests. One for the search term “redux” and another one for “react”. The results object should look like the following:

**Code Playground**

results: {

redux: {

hits: [ ... ],

page: 2,

},

react: {

hits: [ ... ],

page: 1,

},

...

}

Let’s implement a client-side cache with React setState(). First, rename the result object to results in the initial component state. Second, define a temporary searchKey which is used to store each result.

|  |  |
| --- | --- |
| Getting Real with an API | 106 |

**src/App.js**

class App extends Component {

constructor(props) {

super(props);

this.state = {

**results: null,**

**searchKey: '',**

searchTerm: DEFAULT\_QUERY,

};

...

}

...

}

The searchKey has to be set before each request is made. It reflects the searchTerm. You might wonder: Why don’t we use the searchTerm in the first place? That’s a crucial part to understand before continuing with the implementation. The searchTerm is a fluctuant variable, because it gets changed every time you type into the Search input field. However, in the end you will need a non fluctuant variable. It determines the recent submitted search term to the API and can be used to retrieve the correct result from the map of results. It is a pointer to your current result in the cache and thus can be used to display the current result in your render() method.

**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();

}

|  |  |
| --- | --- |
| Getting Real with an API | 107 |

Now you have to adjust the functionality where the result is stored to the internal component state.

It should store each result by searchKey.

**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 }**

**}**

});

}

...

}

The searchKey will be used as the key to save the updated hits and page in a results map.

First, you have to retrieve the searchKey from the component state. Remember that the searchKey gets set on componentDidMount() and onSearchSubmit().

Second, the old hits have to get merged with the new hits as before. But this time the old hits get retrieved from the results map with the searchKey as key.

Third, a new result can be set in the results map in the state. Let’s examine the results object in setState().

|  |  |
| --- | --- |
| Getting Real with an API | 108 |

**src/App.js**

results: {

...results,

[searchKey]: { hits: updatedHits, page }

}

The bottom part makes sure to store the updated result by searchKey in the results map. The value is an object with a hits and page property. The searchKey is the search term. You already learned the [searchKey]: ... syntax. It is an ES6 computed property name. It helps you to allocate values dynamically in an object.

The upper part needs to spread all other results by searchKey in the state by using the object spread operator. Otherwise you would lose all results that you have stored before.

Now you store all results by search term. That’s the first step to enable your cache. In the next step, you can retrieve the result depending on the non fluctuant searchKey from your map of results. That’s why you had to introduce the searchKey in the first place as non fluctuant variable. Otherwise the retrieval would be broken when you would use the fluctuant searchTerm to retrieve the current result, because this value might change when you would use the Search component.

**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**

|  |  |
| --- | --- |
| Getting Real with an API | 109 |

**) || [];**

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>

);

}

}

Since you default to an empty list when there is no result by searchKey, you can spare the conditional rendering for the Table component now. Additionally you will need to pass the searchKey rather than the searchTerm to the “More” button. Otherwise your paginated fetch depends on the searchTerm value which is fluctuant. Moreover make sure to keep the fluctuant searchTerm property for the input field in the “Search” component.

The search functionality should work again. It stores all results from the Hacker News API.

Additionally the onDismiss() method needs to get improved. It still deals with the result object.

Now it has to deal with multiple results.

**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: {**

|  |  |
| --- | --- |
| Getting Real with an API | 110 |

**...results,**

**[searchKey]: { hits: updatedHits, page }**

**}**

**});**

}

The “Dismiss” button should work again.

However, nothing stops the application from sending an API request on each search submit. Even though there might be already a result, there is no check that prevents the request. Thus the cache functionality is not complete yet. It caches the results, but it doesn’t make use of them. The last step would be to prevent the API request when a result is available in the cache.

**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);**

**}**

|  |  |
| --- | --- |
| Getting Real with an API | 111 |

event.preventDefault();

}

...

}

Now your client makes a request to the API only once although you search for a search term twice. Even paginated data with several pages gets cached that way, because you always save the last page for each result in the results map. Isn’t that a powerful approach to introduce caching to your application? The Hacker News API provides you with everything you need to even cache paginated data effectively.

|  |  |
| --- | --- |
| Getting Real with an API | 112 |

**Error Handling**

Everything is in place for your interactions with the Hacker News API. You even have introduced an elegant way to cache your results from the API and make use of its paginated list functionality to fetch an endless list of sublists of stories from the API. But there is one piece missing. Unfortunately it is often missed when developing applications nowadays: error handling. It is too easy to implement the happy path without worrying about the errors that can happen along the way.

In this chapter, you will introduce an efficient solution to add error handling for your application in case of an erroneous API request. You have already learned about the necessary building blocks in React to introduce error handling: local state and conditional rendering. Basically, the error is only another state in React. When an error occurs, you will store it in the local state and display it with a conditional rendering in your component. That’s it. Let’s implement it in the App component, because it’s the component that is used to fetch the data from the Hacker News API in the first place. First, you have to introduce the error in the local state. It is initialized as null, but will be set to the error object in case of an error.

**src/App.js**

class App extends Component {

constructor(props) {

super(props);

this.state = {

results: null,

searchKey: '',

searchTerm: DEFAULT\_QUERY,

**error: null,**

};

...

}

...

}

Second, you can use the catch block in your native fetch to store the error object in the local state by using setState(). Every time the API request isn’t successful, the catch block would be executed.

|  |  |
| --- | --- |
| Getting Real with an API | 113 |

**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(e => this.setState({ error: e }));**

}

...

}

Third, you can retrieve the error object from your local state in the render() method and display a message in case of an error by using React’s conditional rendering.

**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">

|  |  |
| --- | --- |
| Getting Real with an API | 114 |

...

</div>

);

}

}

That’s it. If you want to test that your error handling is working, you can change the API URL to something else that is non existent.

**src/App.js**

**const** PATH\_BASE='https://hn.foo.bar.com/api/v1';

Afterward, you should get the error message instead of your application. It is up to you where you want to place the conditional rendering for the error message. In this case, the whole app isn’t displayed anymore. That wouldn’t be the best user experience. So what about displaying either the Table component or the error message? The remaining application would still be visible in case of an error.

**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

|  |  |
| --- | --- |
| Getting Real with an API | 115 |

) || [];

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>

);

}

}

In the end, don’t forget to revert the URL for the API to the existent one.

**src/App.js**

**const** PATH\_BASE='https://hn.algolia.com/api/v1';

Your application should still work, but this time with error handling in case the API request fails.

**Exercises:**

• read more about [React’s Error Handling for Components¹⁰²](https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html)

¹⁰²<https://reactjs.org/blog/2017/07/26/error-handling-in-react-16.html>

|  |  |
| --- | --- |
| Getting Real with an API | 116 |

You have learned to interact with an API in React! Let’s recap the last chapters:

* React

**–** ES6 class component lifecycle methods for different use cases

**–** componentDidMount() for API interactions

**–** conditional renderings

**–** synthetic events on forms

**–** error handling

* ES6

**–** template strings to compose strings

**–** spread operator for immutable data structures

**–** computed property names

* General

**–** Hacker News API interaction

**–** native fetch browser API

**–** client- and server-side search

**–** pagination of data

**–** client-side caching

Again it makes sense to take a break. Internalize the learnings and apply them on your own. You can experiment with the source code you have written so far.

You can find the source code in the [official repository¹⁰³](https://github.com/rwieruch/hackernews-client/tree/4.3).

¹⁰³<https://github.com/rwieruch/hackernews-client/tree/4.3>

**Code Organization and Testing**

The chapter will focus on important topics to keep your code maintainable in a scaling application. You will learn about code organization to embrace best practices when structuring your folders and files. Another aspect you will learn is testing, which is important to keep your code robust. The whole chapter will take a step back from the practical application and explain a couple of these topics for you.

|  |  |
| --- | --- |
| Code Organization and Testing | 118 |

**ES6 Modules: Import and Export**

In JavaScript ES6 you can import and export functionalities from modules. These functionalities can be functions, classes, components, constants and others. Basically everything that you can assign to a variable. The modules can be single files or whole folders with one index file as entry point.

In the beginning of the book, after you have bootstrapped your application with create-react-app, you already had several importand exportstatements across your initial files. Now it is the appropriate time to explain these.

The import and export statements help you to share code across multiple files. Before there were already several solutions for this in the JavaScript environment. It was a mess, because you would want to follow one standardized way rather than having several approaches for the same thing. Now it is a native behavior since JavaScript ES6.

Additionally these statements embrace code splitting. You distribute your code across multiple files to keep it reusable and maintainable. The former is true because you can import the piece of code in multiple files. The latter is true because you have one single source where you maintain the piece of code.

Last but not least, it helps you to think about code encapsulation. Not every functionality needs to get exported from a file. Some of these functionalities should only be used in the file where they have been defined. The exports of a file are basically the public API to the file. Only the exported functionalities are available to be reused somewhere else. It follows the best practice of encapsulation.

But let’s get practical. How do these import and export statements work? The following examples showcase the statements by sharing one or multiple variables across two files. In the end, the approach can scale to multiple files and could share more than simple variables.

You can export one or multiple variables. It is called a named export.

**Code Playground: file1.js**

**const** firstname='robin';

**const** lastname='wieruch';

**export** { firstname, lastname };

And import them in another file with a relative path to the first file.

|  |  |
| --- | --- |
| Code Organization and Testing | 119 |

**Code Playground: file2.js**

**import** { firstname, lastname } from'./file1.js';

console.log(firstname);

*// output: robin*

You can also import all exported variables from another file as one object.

**Code Playground: file2.js**

**import** \*as person from'./file1.js';

console.log(person.firstname);

*// output: robin*

Imports can have an alias. It can happen that you import functionalities from multiple files that have the same named export. That’s why you can use an alias.

**Code Playground: file2.js**

**import** { firstname as foo } from'./file1.js';

console.log(foo);

*// output: robin*

Last but not least there exists the default statement. It can be used for a few use cases:

* to export and import a single functionality
* to highlight the main functionality of the exported API of a module
* to have a fallback import functionality

|  |  |
| --- | --- |
| Code Organization and Testing | 120 |

**Code Playground: file1.js**

**const** robin={

firstname: 'robin',

lastname: 'wieruch',

};

**export default** robin;

You can leave out the curly braces for the import to import the default export.

**Code Playground: file2.js**

**import** developer from'./file1.js';

console.log(developer);

*// output: { firstname: 'robin', lastname: 'wieruch' }*

Furthermore, the import name can differ from the exported default name. You can also use it in conjunction with the named export and import statements.

**Code Playground: file1.js**

**const** firstname='robin';

**const** lastname='wieruch';

**const** person={

firstname,

lastname,

};

**export** {

firstname,

lastname,

};

**export default** person;

|  |  |
| --- | --- |
| Code Organization and Testing | 121 |

**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*

In named exports you can spare additional lines and export the variables directly.

**Code Playground: file1.js**

**export const** firstname='robin';

**export const** lastname='wieruch';

These are the main functionalities for ES6 modules. They help you to organize your code, to maintain your code and to design reusable module APIs. You can also export and import functionalities to test them. You will do that in one of the following chapters.

**Exercises:**

* read more about [ES6 import¹⁰⁴](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import)
* read more about [ES6 export¹⁰⁵](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export)

¹⁰⁴<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import> ¹⁰⁵<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export>

|  |  |
| --- | --- |
| Code Organization and Testing | 122 |

**Code Organization with ES6 Modules**

You might wonder: Why didn’t we follow the best practices of code splitting for the src/App.js file? In the file we already have multiple components which could be defined in their own files/folders (modules). For the sake of learning React, it is practical to keep these things at one place. But once your React application grows, you should consider to split up these components into multiple modules. Only that way your application scales.

In the following, I will propose several module structures you could apply. I would recommend to apply them as an exercise at the end of the book. To keep the book itself simple, I will not perform the code splitting and will continue the following chapters with the src/App.js file.

One possible module structure could be:

**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

It separates the components into their own files, but it doesn’t look too promising. You can see a lot of naming duplications and only the file extension differs. Another module structure could be:

|  |  |
| --- | --- |
| Code Organization and Testing | 123 |

**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

It looks cleaner than before. The index naming of a file describes it as an entry point file to the folder. It is just a common naming convention, but you can use your own naming as well. In this module structure, a component is defined by its component declaration in the JavasScript file, but also by its style and tests.

Another step could be extracting the constant variables from the App component. These constants were used to compose the Hacker News API URL.

**Folder Structure**

src/

index.js

index.css

**constants/**

**index.js**

**components/**

App/

index.js

test.js

index..css

Button/

index.js

|  |  |
| --- | --- |
| Code Organization and Testing | 124 |

test.js

index..css

...

Naturally the modules would split up into src/constants/ and src/components/. Now the src/con-stants/index.js file could look like the following:

**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=';

The App/index.js file could import these variables in order to use them.

**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';

...

When you use the index.js naming convention, you can omit the filename from the relative path.

|  |  |
| --- | --- |
| Code Organization and Testing | 125 |

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

import {

DEFAULT\_QUERY,

DEFAULT\_HPP,

PATH\_BASE,

PATH\_SEARCH,

PARAM\_SEARCH,

PARAM\_PAGE,

PARAM\_HPP,

**} from '../constants';**

...

But what’s behind the index.js file naming? The convention was introduced in the node.js world. The index file is the entry point to a module. It describes the public API to the module. External modules are only allowed to use the index.js file to import shared code from the module. Consider the following made up module structure to demonstrate it:

**Folder Structure**

src/

index.js

App/

index.js

Buttons/

index.js

SubmitButton.js

SaveButton.js

CancelButton.js

The Buttons/ folder has multiple button components defined in its distinct files. Each file can export default the specific component making it available to *Buttons/index.js*. The *Buttons/index.js* file imports all different button representations and exports them as public module API.

|  |  |
| --- | --- |
| Code Organization and Testing | 126 |

**Code Playground: src/Buttons/index.js**

**import** SubmitButton from'./SubmitButton';

**import** SaveButton from'./SaveButton';

**import** CancelButton from'./CancelButton';

**export** {

SubmitButton,

SaveButton,

CancelButton,

};

Now the src/App/index.js can import the buttons from the public module API located in the index.js file.

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

**import** {

SubmitButton,

SaveButton,

CancelButton

} from '../Buttons';

By going with this constraint, it would be a bad practice to reach into other files than the index.js in the module. It would break the rules of encapsulation.

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

*// bad practice, don't do it*

**import** SubmitButton from'../Buttons/SubmitButton';

Now you know how you could refactor your source code in modules with the constraints of encapsulation. As I said, for the sake of keeping the book simple I will not apply these changes. But you should do the refactoring when you finished reading the book.

**Exercises:**

• refactor your src/App.js file into multiple component modules when you finished the book

|  |  |
| --- | --- |
| Code Organization and Testing | 127 |

**Snapshot Tests with Jest**

The book will not dive deeply into the topic of testing, but it shouldn’t be unmentioned. Testing your code in programming is essential and should be seen as mandatory. You want to keep the quality of your code high and an assurance that everything works.

Perhaps you have heard about the testing pyramid. There are end-to-end tests, integration tests and unit tests. If you are not familiar with those, the book gives you a quick and basic overview. A unit test is used to test an isolated and small block of code. It can be a single function that is tested by an unit test. However, sometimes the units work well in isolation yet don’t work in combination with other units. They need to be tested as a group as units. That’s where integration tests can help out by covering whether units work well together. Last but not least, an end-to-end test is the simulation of a real user scenario. It could be an automated setup in a browser simulating the login flow of an user in a web application. While unit tests are fast and easy to write and to maintain, end-to-end tests are the opposite of this spectrum.

How many tests do I need of each type? You want to have many unit tests to cover your isolated functions. After that, you can have several integration tests to cover that the most important functions work in combination as expected. Last but not least, you might want to have only a few end-to-end tests to simulate critical scenarios in your web application. That’s it for the general excursion in the world of testing.

So how do you apply this knowledge in testing your React application? The foundation for testing in React are component tests which can be generalized as unit tests and a part of it as snapshot tests. You will conduct unit tests for your components in the next chapter by using a library called Enzyme. In this chapter, you will focus on another kind of tests: snapshot tests. That’s were Jest comes into play.

[Jest¹⁰⁶](https://facebook.github.io/jest/) is a JavaScript testing framework that is used at Facebook. In the React community, it is used for React component tests. Fortunately create-react-app already comes with Jest, so you don’t need to worry about setting it up.

Let’s start to test your first components. Before you can do that, you have to export the components, which you are going to test, from your src/App.js file. Afterward you can test them in a different file. You have learned about this in the code organization chapter.

¹⁰⁶<https://facebook.github.io/jest/>

|  |  |
| --- | --- |
| Code Organization and Testing | 128 |

**src/App.js**

...

class App extends Component {

...

}

...

export default App;

**export {**

**Button,**

**Search,**

**Table,**

**};**

In your App.test.js file, you will find a first test that came with create-react-app. It verifies that the App component would render without any errors.

**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);

});

The “it”-block describes one test case. It comes with a test description and when you test it, it can either succeed or fail. Furthermore, you could wrap it into a “describe”-block that defines your test suite. A test suite could include a bunch of the “it”-blocks for one specific component. You will see those “describe”-blocks later on. Both blocks are used to separated and organize your test cases.

Note that the it function is acknowledged in the JavaScript community as the function where you run a single test. However, in Jest it is often found as an alias test function.

You can run your test cases by using the interactive create-react-app test script on the command line. You will get the output for all test cases on your command line interface.

|  |  |
| --- | --- |
| Code Organization and Testing | 129 |

**Command Line**

npm test

**Note:** If errors show up when you run the single test for the App component for the first time, it could be because of the unsupported fetch method that is used in fetchSearchTopStories() which is triggered in componentDidMount(). You can make it work by following these two steps:

* On the command line, install the following package: npm install isomorphic-fetch
* Include it in your App.js file: import fetch from 'isomorphic-fetch';

Now Jest enables you to write snapshot tests. These tests make a snapshot of your rendered component and run this snapshot against future snapshots. When a future snapshot changes, you will get notified in the test. You can either accept the snapshot change, because you changed the component implementation on purpose, or deny the change and investigate for the error. It complements unit tests very well, because you only test the diffs of the rendered output. It doesn’t add big maintenance costs, because you can simply accept changed snapshots when you changed something on purpose for the rendered output in your component.

Jest stores the snapshots in a folder. Only that way it can validate the diff against a future snapshot.

Additionally, the snapshots can be shared across teams by having them in one folder.

Before writing your first snapshot test with Jest, you have to install an utility library.

**Command Line**

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

Now you can extend the App component test with your first snapshot test. First, import the new functionality from the node package and wrap your previous “it”-block for the App component into a descriptive “describe”-block. In this case, the test suite is only for the App component.

**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);

|  |  |
| --- | --- |
| Code Organization and Testing | 130 |

});

**});**

Now you can implement your first snapshot test by using a “test”-block.

**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);

});

**test('has a valid snapshot', () => {**

**const component = renderer.create(**

**<App />**

**);**

**let tree = component.toJSON();**

**expect(tree).toMatchSnapshot();**

**});**

});

Run your tests again and see how the tests either succeed or fail. They should succeed. Once you change the output of the render block in your App component, the snapshot test should fail. Then you can decide to update the snapshot or investigate in your App component.

Basically the renderer.create() function creates a snapshot of your App component. It renders it virtually and stores the DOM into a snapshot. Afterward, the snapshot is expected to match the previous snapshot from when you ran your snapshot tests the last time. This way, you can assure that your DOM stays the same and doesn’t change anything by accident.

Let’s add more tests for our independent components. First, the Search component:

|  |  |
| --- | --- |
| Code Organization and Testing | 131 |

**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);**

**});**

**test('has a valid snapshot', () => {**

**const component = renderer.create(**

**<Search>Search</Search>**

**);**

**let tree = component.toJSON();**

**expect(tree).toMatchSnapshot();**

**});**

**});**

The Search component has two tests similar to the App component. The first test simply renders the Search component to the DOM and verifies that there is no error during the rendering process. If there would be an error, the test would break even though there isn’t any assertion (e.g. expect, match, equal) in the test block. The second snapshot test is used to store a snapshot of the rendered component and to run it against a previous snapshot. It fails when the snapshot has changed.

Second, you can test the Button component whereas the same test rules as in the Search component apply.

|  |  |
| --- | --- |
| Code Organization and Testing | 132 |

**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);**

**});**

**test('has a valid snapshot', () => {**

**const component = renderer.create(**

**<Button>Give Me More</Button>**

**);**

**let tree = component.toJSON();**

**expect(tree).toMatchSnapshot();**

**});**

**});**

Last but not least, the Table component that you can pass a bunch of initial props to render it with a sample list.

**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' }, ],**

**};**

|  |  |
| --- | --- |
| Code Organization and Testing | 133 |

**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();**

**});**

**});**

Snapshot tests usually stay pretty basic. You only want to cover that the component doesn’t change its output. Once it changes the output, you have to decide if you accept the changes. Otherwise you have to fix the component when the output is not the desired output.

**Exercises:**

* see how a snapshot test fails once you change your component’s return value in the render() method

**–** either accept or deny the snapshot change

* keep your snapshots tests up to date when the implementation of components change in next chapters
* read more about [Jest in React¹⁰⁷](https://facebook.github.io/jest/docs/tutorial-react.html)

¹⁰⁷<https://facebook.github.io/jest/docs/tutorial-react.html>

|  |  |
| --- | --- |
| Code Organization and Testing | 134 |

**Unit Tests with Enzyme**

[Enzyme¹⁰⁸](https://github.com/airbnb/enzyme) is a testing utility by Airbnb to assert, manipulate and traverse your React components.

You can use it to conduct unit tests to complement your snapshot tests in React.

Let’s see how you can use enzyme. First you have to install it since it doesn’t come by default with create-react-app. It comes also with an extension to use it in React.

**Command Line**

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

Second, you need to include it in your test setup and initialize its Adapter for using it in React.

**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() });**

Now you can write your first unit test in the Table “describe”-block. You will use shallow() to render your component and assert that the Table has two items, because you pass it two list items. The assertion simply checks if the element has two elements with the class table-row.

**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', () => {

¹⁰⁸<https://github.com/airbnb/enzyme>

|  |  |
| --- | --- |
| Code Organization and Testing | 135 |

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('shows two items in list', () => {**

**const element = shallow(**

**<Table { ...props } />**

**);**

**expect(element.find('.table-row').length).toBe(2); });**

});

Shallow renders the component without its child components. That way, you can make the test very dedicated to one component.

Enzyme has overall three rendering mechanisms in its API. You already know shallow(), but there also exist mount() and render(). Both instantiate instances of the parent component and all child components. Additionally mount() gives you access to the component lifecycle methods. But when to use which render mechanism? Here some rules of thumb:

* Always begin with a shallow test
* If componentDidMount() or componentDidUpdate() should be tested, use mount()
* If you want to test component lifecycle and children behavior, use mount()
* If you want to test a component’s children rendering with less overhead than mount() and you are not interested in lifecycle methods, use render()

You could continue to unit test your components. But make sure to keep the tests simple and maintainable. Otherwise you will have to refactor them once you change your components. That’s why Facebook introduced snapshot tests with Jest in the first place.

**Exercises:**

* write a unit test with Enzyme for your Button component
* keep your unit tests up to date during the following chapters
* read more about [enzyme and its rendering API¹⁰⁹](https://github.com/airbnb/enzyme)

¹⁰⁹<https://github.com/airbnb/enzyme>

|  |  |
| --- | --- |
| Code Organization and Testing | 136 |

**Component Interface with PropTypes**

You may know [TypeScript¹¹⁰](https://www.typescriptlang.org/) or [Flow¹¹¹](https://flowtype.org/) to introduce a type interface to JavaScript. A typed language is less error prone, because the code gets validated based on its program text. Editors and other utilities can catch these errors before the program runs. It makes your program more robust.

In the book, you will not introduce Flow or TypeScript, but another neat way to check your types in components. React comes with a built-in type checker to prevent bugs. You can use PropTypes to describe your component interface. All the props that get passed from a parent component to a child component get validated based on the PropTypes interface assigned to the child component.

The chapter will show you how you can make all your components type safe with PropTypes. I will omit the changes for the following chapters, because they add unnecessary code refactorings. But you should keep and update them along the way to keep your components interface type safe.

First, you have to install a separate package for React.

**Command Line**

npm install prop-types

Now, you can import the PropTypes.

**src/App.js**

**import PropTypes from 'prop-types';**

Let’s start to assign a props interface to the components:

**src/App.js**

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

onClick={onClick}

className={className}

type="button"

>

{children}

</button>

**Button.propTypes = {**

**onClick: PropTypes.func,**

¹¹⁰<https://www.typescriptlang.org/>

¹¹¹<https://flowtype.org/>

|  |  |
| --- | --- |
| Code Organization and Testing | 137 |

**className: PropTypes.string,**

**children: PropTypes.node,**

**};**

Basically that’s it. You take every argument from the function signature and assign a PropType to it. The basic PropTypes for primitives and complex objects are:

* PropTypes.array
* PropTypes.bool
* PropTypes.func
* PropTypes.number
* PropTypes.object
* PropTypes.string

Additionally you have two more PropTypes to define a renderable fragment (node), e.g. a string, and a React element:

* PropTypes.node
* PropTypes.element

You already used the node PropType for the Button component. Overall there are more PropType definitions that you can read up on in the official React documentation.

At the moment all of the defined PropTypes for the Button are optional. The parameters can be null or undefined. But for several props you want to enforce that they are defined. You can make it a requirement that these props are passed to the component.

**src/App.js**

Button.propTypes = {

**onClick: PropTypes.func.isRequired,**

className: PropTypes.string,

**children: PropTypes.node.isRequired,**

};

The className is not required, because it can default to an empty string. Next you will define a

PropType interface for the Table component:

|  |  |
| --- | --- |
| Code Organization and Testing | 138 |

**src/App.js**

**Table.propTypes = {**

**list: PropTypes.array.isRequired,**

**onDismiss: PropTypes.func.isRequired,**

**};**

You can define the content of an array PropType more explicitly:

**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,

};

Only the objectID is required, because you know that some of your code depends on it. The other properties are only displayed, thus they are not necessarily required. Moreover you cannot be sure that the Hacker News API has always a defined property for each object in the array.

That’s it for PropTypes. But there is one more aspect. You can define default props in your component. Let’s take again the Button component. The className property has an ES6 default parameter in the component signature.

**src/App.js**

**const** Button=({

onClick,

className = '',

children

}) =>

...

You could replace it with the internal React default prop:

|  |  |
| --- | --- |
| Code Organization and Testing | 139 |

**src/App.js**

**const Button = ({**

**onClick,**

**className,**

**children**

**}) =>**

<button

onClick={onClick}

className={className}

type="button"

>

{children}

</button>

**Button.defaultProps = {**

**className: '',**

**};**

Same as the ES6 default parameter, the default prop ensures that the property is set to a default value when the parent component didn’t specify it. The PropType type check happens after the default prop is evaluated.

If you run your tests again, you might see PropType errors for your components on your command line. It can happen because you didn’t define all props for your components in the tests that are defined as required in your PropType definition. The tests themselves all pass correctly though. You can pass all required props to avoid these errors.

**Exercises:**

* define the PropType interface for the Search component
* add and update the PropType interfaces when you add and update components in the next chapters
* read more about [React PropTypes¹¹²](https://facebook.github.io/react/docs/typechecking-with-proptypes.html)

¹¹²<https://facebook.github.io/react/docs/typechecking-with-proptypes.html>

|  |  |
| --- | --- |
| Code Organization and Testing | 140 |

You have learned how to organize your code and how to test it! Let’s recap the last chapters:

* React

**–** PropTypes let you define type checks for components

**–** Jest allows you to write snapshot tests for your components

**–** Enzyme allows you to write unit tests for your components

* ES6

**–** import and export statements help you to organize your code

* General

**–** code organization allows you to scale your application with best practices

You can find the source code in the [official repository¹¹³](https://github.com/rwieruch/hackernews-client/tree/4.4).

¹¹³<https://github.com/rwieruch/hackernews-client/tree/4.4>

**Advanced React Components**

The chapter will focus on the implementation of advanced React components. You will learn about higher-order components and how to implement them. In addition you will dive into more advanced topics in React and implement complex interactions with it.

|  |  |
| --- | --- |
| Advanced React Components | 142 |

**Ref a DOM Element**

Sometimes you need to interact with your DOM nodes in React. The ref attribute gives you access to a node in your elements. Usually that is an anti pattern in React, because you should use its declarative way of doing things and its unidirectional data flow. You have learned about it when you have introduced your first search input field. But there are certain cases where you need access to the DOM node. The official documentation mentions three use cases:

* to use the DOM API (focus, media playback etc.)
* to invoke imperative DOM node animations
* to integrate with a third-party library that needs the DOM node (e.g. [D3.js¹¹⁴](https://d3js.org/))

Let’s do it by example with the Search component. When the application renders the first time, the input field should be focused. That’s one use case where you would need access to the DOM API. This chapter will show you how it works, but since it is not very useful for the application itself, we will omit the changes after the chapter. You can keep it for your own application though.

In general, you can use the ref attribute in both functional stateless components and ES6 class components. In the example of the focus use case, you will need a lifecycle method. That’s why the approach is first showcased by using the ref attribute with an ES6 class component.

The initial step is to refactor the functional stateless component to an ES6 class component.

**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">

¹¹⁴<https://d3js.org/>

|  |  |
| --- | --- |
| Advanced React Components | 143 |

{children}

</button>

</form>

**);**

**}**

**}**

The this object of an ES6 class component helps us to reference the DOM node with the ref attribute.

**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>

);

}

}

Now you can focus the input field when the component mounted by using the this object, the appropriate lifecycle method, and the DOM API.

|  |  |
| --- | --- |
| Advanced React Components | 144 |

**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>

);

}

}

The input field should be focused when the application renders. That’s it basically for using the ref attribute.

But how would you get access to the ref in a functional stateless component without the this object? The following functional stateless component demonstrates it.

|  |  |
| --- | --- |
| Advanced React Components | 145 |

**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>

);

}

Now you would be able to access the input DOM element. In the example of the focus use case it wouldn’t help you, because you have no lifecycle method in a functional stateless component to trigger the focus. But in the future you might come across other use cases where it can make sense to use a functional stateless component with the ref attribute.

**Exercises**

* read more about [the ref attribute in general in React¹¹⁵](https://facebook.github.io/react/docs/refs-and-the-dom.html)
* read more about [the usage of the ref attribute in React¹¹⁶](https://www.robinwieruch.de/react-ref-attribute-dom-node/)

¹¹⁵<https://facebook.github.io/react/docs/refs-and-the-dom.html>

¹¹⁶<https://www.robinwieruch.de/react-ref-attribute-dom-node/>

|  |  |
| --- | --- |
| Advanced React Components | 146 |

**Loading …**

Now let’s get back to the application. You might want to show a loading indicator when you submit a search request to the Hacker News API. The request is asynchronous and you should show your user some feedback that something is about to happen. Let’s define a reusable Loading component in your src/App.js file.

**src/App.js**

**const** Loading=() =>

<div>Loading ...</div>

Now you will need a property to store the loading state. Based on the loading state you can decide to show the Loading component later on.

**src/App.js**

class App extends Component {

constructor(props) {

super(props);

this.state = {

results: null,

searchKey: '',

searchTerm: DEFAULT\_QUERY,

error: null,

**isLoading: false,**

};

...

}

...

}

The initial value of that isLoading property is false. You don’t load anything before the App component is mounted.

When you make the request, you set a loading state to true. Eventually the request will succeed and you can set the loading state to false.

|  |  |
| --- | --- |
| Advanced React Components | 147 |

**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 });**

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(e => this.setState({ error: e }));

}

...

}

In the last step, you will use the Loading component in your App. A conditional rendering based on the loading state will decide whether you show a Loading component or the Button component. The latter one is your button to fetch more data.

|  |  |
| --- | --- |
| Advanced React Components | 148 |

**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>

);

}

}

Initially the Loading component will show up when you start your application, because you make a request on componentDidMount(). There is no Table component, because the list is empty. When the response returns from the Hacker News API, the result is shown, the loading state is set to false and the Loading component disappears. Instead, the “More” button to fetch more data appears. Once you fetch more data, the button will disappear again and the Loading component will show up.

|  |  |
| --- | --- |
| Advanced React Components | 149 |

**Exercises:**

• use a library such as [Font Awesome¹¹⁷](http://fontawesome.io/) to show a loading icon instead of the “Loading …” text

¹¹⁷<http://fontawesome.io/>

|  |  |
| --- | --- |
| Advanced React Components | 150 |

**Higher-Order Components**

Higher-order components (HOC) are an advanced concept in React. HOCs are an equivalent to higher-order functions. They take any input - most of the time a component, but also optional arguments - and return a component as output. The returned component is an enhanced version of the input component and can be used in your JSX.

HOCs are used for different use cases. They can prepare properties, manage state or alter the representation of a component. One use case could be to use a HOC as a helper for a conditional rendering. Imagine you have a List component that renders a list of items or nothing, because the list is empty or null. The HOC could shield away that the list would render nothing when there is no list. On the other hand, the plain List component doesn’t need to bother anymore about an non existent list. It only cares about rendering the list.

Let’s do a simple HOC which takes a component as input and returns a component. You can place it in your src/App.js file.

**src/App.js**

**function** withFoo(Component) {

**return function**(props) {

**return** <Component { ...props }/>;

}

}

One neat convention is to prefix the naming of a HOC with with. Since you are using JavaScript ES6, you can express the HOC more concisely with an ES6 arrow function.

**src/App.js**

**const** withFoo=(Component) => (props) =>

<Component { ...props } />

In the example, the input component would stay the same as the output component. Nothing happens. It renders the same component instance and passes all of the props to the output component. But that’s useless. Let’s enhance the output component. The output component should show the Loading component, when the loading state is true, otherwise it should show the input component. A conditional rendering is a great use case for a HOC.

|  |  |
| --- | --- |
| Advanced React Components | 151 |

**src/App.js**

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

**? <Loading />**

**: <Component { ...props } />**

Based on the loading property you can apply a conditional rendering. The function will return the Loading component or the input component.

In general it can be very efficient to spread an object, like the props object in the previous example, as input for a component. See the difference in the following code snippet.

**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 } />

There is one little thing that you should avoid. You pass all the props including the isLoading property, by spreading the object, into the input component. However, the input component may not care about the isLoading property. You can use the ES6 rest destructuring to avoid it.

**src/App.js**

**const withLoading = (Component) => ({ isLoading, ...rest }) => isLoading**

**? <Loading />**

**: <Component { ...rest } />**

It takes one property out of the object, but keeps the remaining object. It works with multiple properties as well. You might have already read about it in the [destructuring assignment¹¹⁸](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment).

Now you can use the HOC in your JSX. An use case in the application could be to show either the “More” button or the Loading component. The Loading component is already encapsulated in the HOC, but an input component is missing. In the use case of showing a Button component or a Loading component, the Button is the input component of the HOC. The enhanced output component is a ButtonWithLoading component.

¹¹⁸<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>

|  |  |
| --- | --- |
| Advanced React Components | 152 |

**src/App.js**

const Button = ({ onClick, className = '', 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);**

Everything is defined now. As a last step, you have to use the ButtonWithLoading component, which receives the loading state as an additional property. While the HOC consumes the loading property, all other props get passed to the Button component.

**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>

|  |  |
| --- | --- |
| Advanced React Components | 153 |

</div>

);

}

}

When you run your tests again, you will notice that your snapshot test for the App component fails.

The diff might look like the following on the command line:

**Command Line**

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

You can either fix the component now, when you think there is something wrong about it, or can accept the new snapshot of it. Because you introduced the Loading component in this chapter, you can accept the altered snapshot test on the command line in the interactive test.

Higher-order components are an advanced technique in React. They have multiple purposes like improved reusability of components, greater abstraction, composability of components and manipulations of props, state and view. Don’t worry if you don’t understand them immediately. It takes time to get used to them.

I encourage you to read the [gentle introduction to higher-order components¹¹⁹](https://www.robinwieruch.de/gentle-introduction-higher-order-components/). It gives you another approach to learn them, shows you an elegant way to use them in a functional programming way and solves specifically the problem of conditional rendering with higher-order components.

**Exercises:**

* read [a gentle introduction to higher-order components¹²⁰](https://www.robinwieruch.de/gentle-introduction-higher-order-components/)
* experiment with the HOC you have created
* think about a use case where another HOC would make sense

**–** implement the HOC, if there is a use case

¹¹⁹<https://www.robinwieruch.de/gentle-introduction-higher-order-components/> ¹²⁰<https://www.robinwieruch.de/gentle-introduction-higher-order-components/>

|  |  |
| --- | --- |
| Advanced React Components | 154 |

**Advanced Sorting**

You have already implemented a client- and server-side search interaction. Since you have a Table component, it would make sense to enhance the Table with advanced interactions. What about introducing a sort functionality for each column by using the column headers of the Table?

It would be possible to write your own sort function, but personally I prefer to use a utility library for such cases. [Lodash¹²¹](https://lodash.com/) is one of these utility libraries, but you can use whatever library suits you. Let’s install Lodash and use it for the sort functionality.

**Command Line**

npm install lodash

Now you can import the sort functionality of Lodash in your src/App.js file.

**src/App.js**

import React, { Component } from 'react';

import fetch from 'isomorphic-fetch';

**import { sortBy } from 'lodash';**

import './App.css';

You have several columns in your Table. There are title, author, comments and points columns. You can define sort functions whereas each function takes a list and returns a list of items sorted by a specific property. Additionally, you will need one default sort function which doesn’t sort but only returns the unsorted list. That will be your initial state.

**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 {

...

}

...

¹²¹<https://lodash.com/>

|  |  |
| --- | --- |
| Advanced React Components | 155 |

You can see that two of the sort functions return a reversed list. That’s because you want to see the items with the highest comments and points rather than to see the items with the lowest counts when you sort the list for the first time.

The SORTS object allows you to reference any sort function now.

Again your App component is responsible for storing the state of the sort. The initial state will be the initial default sort function, which doesn’t sort at all and returns the input list as output.

**src/App.js**

this.state = {

results: null,

searchKey: '',

searchTerm: DEFAULT\_QUERY,

error: null,

isLoading: false,

**sortKey: 'NONE',**

};

Once you choose a different sortKey, let’s say the AUTHOR key, you will sort the list with the appropriate sort function from the SORTS object.

Now you can define a new class method in your App component that simply sets a sortKey to your local component state. Afterward, the sortKey can be used to retrieve the sorting function to apply it on your list.

**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.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) {**

|  |  |
| --- | --- |
| Advanced React Components | 156 |

**this.setState({ sortKey });**

**}**

...

}

The next step is to pass the method and sortKey to your Table component.

**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>

);

}

}

|  |  |
| --- | --- |
| Advanced React Components | 157 |

The Table component is responsible for sorting the list. It takes one of the SORT functions by sortKey and passes the list as input. Afterward it keeps mapping over the sorted list.

**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>

In theory the list would get sorted by one of the functions. But the default sort is set to NONE, so nothing is sorted yet. So far, no one executes the onSort() method to change the sortKey. Let’s extend the Table with a row of column headers that use Sort components in columns to sort each column.

**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%' }}>**

|  |  |
| --- | --- |
| Advanced React Components | 158 |

**<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>

Each Sort component gets a specific sortKey and the general onSort() function. Internally it calls the method with the sortKey to set the specific key.

**src/App.js**

**const** Sort=({ sortKey, onSort, children }) =><Button onClick={() => onSort(sortKey)}>

{children}

</Button>

|  |  |
| --- | --- |
| Advanced React Components | 159 |

As you can see, the Sort component reuses your common Button component. On a button click each individual passed sortKey will get set by the onSort() method. Now you should be able to sort the list when you click on the column headers.

There is one minor improvement for an improved look. So far, the button in a column header looks a bit silly. Let’s give the button in the Sort component a proper className.

**src/App.js**

const Sort = ({ sortKey, onSort, children }) =>

**<Button**

**onClick={() => onSort(sortKey)}**

**className="button-inline"**

**>**

{children}

</Button>

It should look nice now. The next goal would be to implement a reverse sort as well. The list should get reverse sorted once you click a Sort component twice. First, you need to define the reverse state with a boolean. The sort can be either reversed or non reversed.

**src/App.js**

this.state = {

results: null,

searchKey: '',

searchTerm: DEFAULT\_QUERY,

error: null,

isLoading: false,

sortKey: 'NONE',

**isSortReverse: false,**

};

Now in your sort method, you can evaluate if the list is reverse sorted. It is reverse if the sortKey in the state is the same as the incoming sortKey and the reverse state is not already set to true.

|  |  |
| --- | --- |
| Advanced React Components | 160 |

**src/App.js**

onSort(sortKey) {

**const isSortReverse = this.state.sortKey === sortKey && !this.state.isSortReve\ rse;**

**this.setState({ sortKey, isSortReverse });**

}

Again you can pass the reverse prop to your Table component.

**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>

);

}

}

|  |  |
| --- | --- |
| Advanced React Components | 161 |

The Table has to have an arrow function block body to compute the data now.

**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>

**);**

**}**

The reverse sort should work now.

Last but not least, you have to deal with one open question for the sake of an improved user experience. Can a user distinguish which column is actively sorted? So far, it is not possible. Let’s give the user a visual feedback.

Each Sort component gets its specific sortKey already. It could be used to identify the activated sort. You can pass the sortKey from the internal component state as active sort key to your Sort component.

|  |  |
| --- | --- |
| Advanced React Components | 162 |

**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>

|  |  |
| --- | --- |
| Advanced React Components | 163 |

</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>

);

}

Now in your Sort component, you know based on the sortKey and activeSortKey whether the sort is active. Give your Sort component an extra className attribute, in case it is sorted, to give the user a visual feedback.

**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(' ')}**

|  |  |
| --- | --- |
| Advanced React Components | 164 |

**>**

**{children}**

**</Button>**

**);**

**}**

The way to define the sortClass is a bit clumsy, isn’t it? There is a neat little library to get rid of this. First you have to install it.

**Command Line**

npm install classnames

And second you have to import it on top of your src/App.js file.

**src/App.js**

import React, { Component } from 'react';

import fetch from 'isomorphic-fetch';

import { sortBy } from 'lodash';

**import classNames from 'classnames';**

import './App.css';

Now you can use it to define your component className with conditional classes.

**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}**

**>**

|  |  |
| --- | --- |
| Advanced React Components | 165 |

{children}

</Button>

);

}

Again, when you run your tests, you should see failing snapshot tests but also failing unit tests for the Table component. Since you changed again your component representations, you can accept the snapshot tests. But you have to fix the unit test. In your src/App.test.js file, you need to provide a sortKey and the isSortReverse boolean for the Table component.

**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,**

};

...

});

Once again you might need to accept the failing snapshot tests for your Table component, because you provided extended props for the Table component.

Finally your advanced sort interaction is complete now.

**Exercises:**

* use a library like [Font Awesome¹²²](http://fontawesome.io/) to indicate the (reverse) sort

**–** it could be an arrow up or arrow down icon next to each Sort header

* read more about the [classnames library¹²³](https://github.com/JedWatson/classnames)

¹²²<http://fontawesome.io/>

¹²³<https://github.com/JedWatson/classnames>

|  |  |
| --- | --- |
| Advanced React Components | 166 |

You have learned advanced component techniques in React! Let’s recap the last chapters:

* React

**–** the ref attribute to reference DOM nodes

**–** higher-order components are a common way to build advanced components

**–** implementation of advanced interactions in React

**–** conditional classNames with a neat helper library

* ES6

**–** rest destructuring to split up objects and arrays

You can find the source code in the [official repository¹²⁴](https://github.com/rwieruch/hackernews-client/tree/4.5).

¹²⁴<https://github.com/rwieruch/hackernews-client/tree/4.5>

**State Management in React and beyond**

You have already learned the basics of state management in React in the previous chapters. This chapter digs a bit deeper into the topic. You will learn best practices, how to apply them and why you could consider using a third-party state management library.

|  |  |
| --- | --- |
| State Management in React and beyond | 168 |

**Lifting State**

Only the App component is a stateful ES6 component in your application. It handles a lot of application state and logic in its class methods. Maybe you have noticed that you pass a lot of properties to your Table component. Most of these props are only used in the Table component. In conclusion one could argue that it makes no sense that the App component knows about them.

The whole sort functionality is only used in the Table component. You could move it into the Table component, because the App component doesn’t need to know about it at all. The process of refactoring substate from one component to another is known as lifting state. In your case, you want to move state that isn’t used in the App component into the Table component. The state moves down from parent to child component.

In order to deal with state and class methods in the Table component, it has to become an ES6 class component. The refactoring from functional stateless component to ES6 class component is straightforward.

Your Table component as a functional stateless component:

**src/App.js**

**const** Table=({

list,

sortKey,

isSortReverse,

onSort,

onDismiss

}) => {

**const** sortedList=SORTS[sortKey](list);

**const** reverseSortedList=isSortReverse

* sortedList.reverse() : sortedList;

**return**(

...

);

}

Your Table component as an ES6 class component:

|  |  |
| --- | --- |
| State Management in React and beyond | 169 |

**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(**

**...**

**);**

**}**

**}**

Since you want to deal with state and methods in your component, you have to add a constructor and initial state.

**src/App.js**

class Table extends Component {

**constructor(props) {**

**super(props);**

**this.state = {};**

**}**

render() {

...

}

}

Now you can move state and class methods regarding the sort functionality from your App component down to your Table component.

|  |  |
| --- | --- |
| State Management in React and beyond | 170 |

**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() {

...

}

}

Don’t forget to remove the moved state and onSort() class method from your App component.

**src/App.js**

**class** App **extends** Component {

constructor(props) {

**super**(props);

**this**.state={

results: **null**,

searchKey: '',

searchTerm: DEFAULT\_QUERY,

error: **null**,

isLoading: **false**,

};

**this**.setSearchTopStories= **this**.setSearchTopStories.bind(**this**);

|  |  |
| --- | --- |
| State Management in React and beyond | 171 |

**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**);

}

...

}

Additionally, you can make the Table component API more lightweight. Remove the props that are passed to it from the App component, because they are handled internally in the Table component now.

**src/App.js**

class App extends Component {

...

render() {

**const {**

**searchTerm,**

**results,**

**searchKey,**

**error,**

**isLoading**

**} = this.state;**

...

return (

<div className="page">

...

**<Table**

**list={list}**

**onDismiss={this.onDismiss}**

**/>**

...

</div>

);

|  |  |
| --- | --- |
| State Management in React and beyond | 172 |

}

}

Now in your Table component you can use the internal onSort() method and the internal Table state.

**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

</Sort>

</span>

<span style={{ width: '30%' }}>

<Sort

sortKey={'AUTHOR'}

|  |  |
| --- | --- |
| State Management in React and beyond | 173 |

**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>

);

}

}

Your application should still work. But you made a crucial refactoring. You moved functionality and state closer into another component. Other components got more lightweight again. Additionally the component API of the Table got more lightweight because it deals internally with the sort functionality.

The process of lifting state can go the other way as well: from child to parent component. It is called as lifting state up. Imagine you were dealing with internal state in a child component. Now you

|  |  |
| --- | --- |
| State Management in React and beyond | 174 |

want to fulfill a requirement to show the state in your parent component as well. You would have to lift up the state to your parent component. But it goes even further. Imagine you want to show the state in a sibling component of your child component. Again you would have to lift the state up to your parent component. The parent component deals with the internal state, but exposes it to both child components.

**Exercises:**

* read more about [lifting state in React¹²⁵](https://facebook.github.io/react/docs/lifting-state-up.html)
* read more about lifting state in [learn React before using Redux¹²⁶](https://www.robinwieruch.de/learn-react-before-using-redux/)

¹²⁵<https://facebook.github.io/react/docs/lifting-state-up.html>

¹²⁶<https://www.robinwieruch.de/learn-react-before-using-redux/>

|  |  |
| --- | --- |
| State Management in React and beyond | 175 |

**Revisited: setState()**

So far, you have used React setState() to manage your internal component state. You can pass an object to the function where you can update partially the internal state.

**Code Playground**

**this**.setState({ foo:bar });

But setState() doesn’t take only an object. In its second version, you can pass a function to update the state.

**Code Playground**

**this**.setState((prevState, props) => {

...

});

Why should you want to do that? There is one crucial use case where it makes sense to use a function over an object. It is when you update the state depending on the previous state or props. If you don’t use a function, the internal state management can cause bugs.

But why does it cause bugs to use an object over a function when the update depends on the previous state or props? The React setState() method is asynchronous. React batches setState() calls and executes them eventually. It can happen that the previous state or props changed in between when you would rely on it in your setState() call.

**Code Playground**

**const** { fooCount }= **this**.state;

**const** { barCount }= **this**.props;

**this**.setState({ count:fooCount+barCount });

Imagine that fooCount and barCount, thus the state or the props, change somewhere else asyn-chronously when you call setState(). In a growing application, you have more than one ‘setState()’ call across your application. Since setState() executes asynchronously, you could rely in the example on stale values.

With the function approach, the function in setState() is a callback that operates on the state and props at the time of executing the callback function. Even though setState() is asynchronous, with a function it takes the state and props at the time when it is executed.

|  |  |
| --- | --- |
| State Management in React and beyond | 176 |

**Code Playground**

**this**.setState((prevState, props) => {

**const** { fooCount }=prevState;

**const** { barCount }=props;

**return** { count:fooCount+barCount };

});

Now, lets get back to your code to fix this behavior. Together we will fix it for one place where setState() is used and relies on the state or props. Afterward, you are able to fix it at other places too.

The setSearchTopStories() method relies on the previous state and thus is a perfect example to use a function over an object in setState(). Right now, it looks like the following code snippet.

**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**

});

}

You extract values from the state, but update the state depending on the previous state asyn-chronously. Now you can use the functional approach to prevent bugs because of a stale state.

|  |  |
| --- | --- |
| State Management in React and beyond | 177 |

**src/App.js**

setSearchTopStories(result) {

const { hits, page } = result;

**this.setState(prevState => {**

**...**

**});**

}

You can move the whole block that you have already implemented into the function. You only have to exchange that you operate on the prevState rather than this.state.

**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**

**};**

});

}

That will fix the issue with a stale state. There is one more improvement. Since it is a function, you can extract the function for an improved readability. That’s one more advantage to use a function

|  |  |
| --- | --- |
| State Management in React and beyond | 178 |

over an object. The function can live outside of the component. But you have to use a higher-order function to pass the result to it. After all, you want to update the state based on the fetched result from the API.

**src/App.js**

setSearchTopStories(result) {

**const** { hits, page }=result;

**this**.setState(updateSearchTopStoriesState(hits, page));

}

The updateSearchTopStoriesState() function has to return a function. It is a higher-order function. You can define this higher-order function outside of your App component. Note how the function signature changes slightly now.

**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 {

...

}

That’s it. The function over an object approach in setState() fixes potential bugs yet increases readability and maintainability of your code. Furthermore, it becomes testable outside of the App component. You could export it and write a test for it as exercise.

|  |  |
| --- | --- |
| State Management in React and beyond | 179 |

**Exercise:**

* read more about [React using state correctly¹²⁷](https://facebook.github.io/react/docs/state-and-lifecycle.html#using-state-correctly)
* refactor all setState() methods to use a function

**–** but only when it makes sense, because it relies on props or state

* run your tests again and verify that everything is up to date

¹²⁷<https://facebook.github.io/react/docs/state-and-lifecycle.html#using-state-correctly>

|  |  |
| --- | --- |
| State Management in React and beyond | 180 |

**Taming the State**

The previous chapters have shown you that state management can be a crucial topic in larger applications. In general, not only React but a lot of SPA frameworks struggle with it. Applications got more complex in the recent years. One big challenge in web applications nowadays is to tame and control the state.

Compared to other solutions, React already made a big step forward. The unidirectional data flow and a simple API to manage state in a component are indispensable. These concepts make it easier to reason about your state and your state changes. It makes it easier to reason about it on a component level and to a certain degree on an application level.

In a growing application, it gets harder to reason about state changes. You can introduce bugs by operating on stale state when using an object over a function in setState(). You have to lift state around to share necessary or hide unnecessary state across components. It can happen that a component needs to lift up state, because its sibling component depends on it. Perhaps the component is far away in the component tree and thus you have to share the state across the whole component tree. In conclusion components get involved to a greater extent in state management. But after all, the main responsibility of components should be representing the UI, shouldn’t it?

Because of all these reasons, there exist standalone solutions to take care of the state management. These solutions are not only used in React. However, that’s what makes the React ecosystem such a powerful place. You can use different solutions to solve your problems. To address the problem of scaling state management, you might have heard of the libraries [Redux¹²⁸](http://redux.js.org/docs/introduction/) or [MobX¹²⁹](https://mobx.js.org/). You can use either of these solutions in a React application. They come with extensions, [react-redux¹³⁰](https://github.com/reactjs/react-redux) and [mobx-react¹³¹](https://github.com/mobxjs/mobx-react), to integrate them into the React view layer.

Redux and MobX are outside of the scope of this book. When you have finished the book, you will get guidance on how you can continue to learn React and its ecosystem. One learning path could be to learn Redux. Before you dive into the topic of external state management, I can recommend to read this [article¹³²](https://www.robinwieruch.de/redux-mobx-confusion/). It aims to give you a better understanding of how to learn external state management.

**Exercises:**

* read more about [external state management and how to learn](https://www.robinwieruch.de/redux-mobx-confusion/) it¹³³
* check out my second ebook about [state management in React¹³⁴](https://roadtoreact.com/)

¹²⁸<http://redux.js.org/docs/introduction/>

¹²⁹<https://mobx.js.org/>

¹³⁰<https://github.com/reactjs/react-redux>

¹³¹<https://github.com/mobxjs/mobx-react>

¹³²<https://www.robinwieruch.de/redux-mobx-confusion/>

¹³³<https://www.robinwieruch.de/redux-mobx-confusion/>

¹³⁴<https://roadtoreact.com/>

|  |  |
| --- | --- |
| State Management in React and beyond | 181 |

You have learned advanced state management in React! Let’s recap the last chapters:

* React

**–** lift state management up and down to suitable components

**–** setState can use a function to prevent stale state bugs

**–** existing external solutions that help you to tame the state

You can find the source code in the [official repository¹³⁵](https://github.com/rwieruch/hackernews-client/tree/4.6).

¹³⁵<https://github.com/rwieruch/hackernews-client/tree/4.6>

**Final Steps to Production**

The last chapters will show you how to deploy your application to production. You will use the free hosting service Heroku. On the way to deploy your application, you will learn more about create-react-app.

|  |  |
| --- | --- |
| Final Steps to Production | 183 |

**Eject**

The following step and knowledge is not necessary to deploy your application to production. Still, I want to explain it to you. create-react-app comes with one feature to keep it extendable but also to prevent a vendor lock-in. A vendor lock-in usually happens when you buy into a technology but there is no escape hatch of using it in the future. Fortunately in create-react-app you have such an escape hatch with “eject”.

In your package.json you will find the scripts to start, test and build your application. The last script is eject. You could try it, but there is no way back. It is a one-way operation. Once you eject, you can’t go back! If you just started to learn React, it makes no sense to leave the convenient environment of create-react-app.

If you would run npm run eject, the command would copy all the configuration and dependencies to your package.json and a new config/ folder. You would convert the whole project into a custom setup with tooling that includes Babel and Webpack. After all, you would have full control over all these tools.

The official documentation says that create-react-app is suitable for small to mid size projects. You shouldn’t feel obligated to use the “eject” command.

**Exercises:**

• read more about [eject¹³⁶](https://github.com/facebookincubator/create-react-app#converting-to-a-custom-setup)

¹³⁶<https://github.com/facebookincubator/create-react-app#converting-to-a-custom-setup>

|  |  |
| --- | --- |
| Final Steps to Production | 184 |

**Deploy your App**

In the end, no application should stay on localhost. You want to go live. Heroku is a platform as a service where you can host your application. They offer a seamless integration with React. To be more specific: It’s possible to deploy a create-react-app in minutes. It is a zero-configuration deployment which follows the philosophy of create-react-app.

You need to fulfill two requirements before you can deploy your application to Heroku:

* install the [Heroku CLI¹³⁷](https://devcenter.heroku.com/articles/heroku-command-line)
* create a [free Heroku account¹³⁸](https://www.heroku.com/)

If you have installed Homebrew, you can install the Heroku CLI from command line:

**Command Line**

brew update

brew install heroku-toolbelt

Now you can use git and Heroku CLI to deploy your application.

**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

That’s it. I hope your application is up and running now. If you run into problems you can check the following resources:

* [Git and GitHub Essentials¹³⁹](https://www.robinwieruch.de/git-essential-commands/)
* [Deploying React with Zero Configuration¹⁴⁰](https://blog.heroku.com/deploying-react-with-zero-configuration)
* [Heroku Buildpack for create-react-app¹⁴¹](https://github.com/mars/create-react-app-buildpack)

¹³⁷<https://devcenter.heroku.com/articles/heroku-command-line>

¹³⁸<https://www.heroku.com/>

¹³⁹<https://www.robinwieruch.de/git-essential-commands/>

¹⁴⁰<https://blog.heroku.com/deploying-react-with-zero-configuration>

¹⁴¹<https://github.com/mars/create-react-app-buildpack>

**Outline**

That was the last chapter of the book. I hope you enjoyed reading it and that it helped you to get traction in React. If you liked the book, share it as a way to learn React with your friends. It should be used as giveaway. In addition, you could take 5 minutes to write a quick review about it on [Amazon¹⁴²](https://www.amazon.com/dp/B077HJFCQX?tag=21moves-20).

**But where can you go from here after reading this book?** You can either extend the application on your own or give your own React project a shot. Before you dive into another book, course or tutorial, you should create your own hands-on React project. Do it for one week, take it to production by deploying it somewhere, and reach out to me on [Twitter¹⁴³](https://twitter.com/rwieruch) to showcase it. I am curious what you will build after you have read the book and I will gladly share it with my followers. You can also find me on [GitHub¹⁴⁴](https://github.com/rwieruch) to share your repository.

If you are looking for further extensions for your application, I can recommend several learning paths after you have used only plain React in this book:

* **State Management:** You have used React this.setState()and this.stateto manage and access local component state. That’s a perfect start. However, in a larger application you will experience the [limits of React’s local component state¹⁴⁵](https://www.robinwieruch.de/learn-react-before-using-redux/). Therefore you can use a third-party state management library such as [Redux or MobX¹⁴⁶](https://www.robinwieruch.de/redux-mobx-confusion/). On the course platform [Road to React¹⁴⁷](https://roadtoreact.com/), you will find the course “Taming the State in React” that teaches advanced local state in React, Redux and MobX. The course comes with an ebook as well, but I recommend everyone to dive into the source code and screencasts too. If you liked this book, you should definitely checkout Taming the State in React.
* **Sample Projects:** After learning plain React, it is always good to apply the learnings first in your own projects before learning something new. You could write your own tic-tac-toe game or a simple calculator in React. There are plenty of tutorials out there that use only React to build something exciting. Check out mine about building [a paginated and infinite scrolling](https://www.robinwieruch.de/react-paginated-list/) [list¹⁴⁸](https://www.robinwieruch.de/react-paginated-list/), [showcasing tweets on a Twitter wall¹⁴⁹](https://www.robinwieruch.de/react-svg-patterns/) or [connecting your React application to Stripe](https://www.robinwieruch.de/react-express-stripe-payment/) [for charging money¹⁵⁰](https://www.robinwieruch.de/react-express-stripe-payment/). Experiment with these mini applications to get comfortable in React.

¹⁴²<https://www.amazon.com/dp/B077HJFCQX?tag=21moves-20>

¹⁴³<https://twitter.com/rwieruch>

¹⁴⁴<https://github.com/rwieruch>

¹⁴⁵<https://www.robinwieruch.de/learn-react-before-using-redux/>

¹⁴⁶<https://www.robinwieruch.de/redux-mobx-confusion/>

¹⁴⁷<https://roadtoreact.com/>

¹⁴⁸<https://www.robinwieruch.de/react-paginated-list/>

¹⁴⁹<https://www.robinwieruch.de/react-svg-patterns/>

¹⁵⁰<https://www.robinwieruch.de/react-express-stripe-payment/>

|  |  |
| --- | --- |
| Outline | 186 |

* **Connecting to a Database and/or Authentication:** In a growing React application, you may want to persist data eventually. The data should be stored in a database so that it can survive after a browser session and be shared across different users using your application. The simplest way to introduce a database is using Firebase. In [this comprehensive tutorial¹⁵¹](https://www.robinwieruch.de/complete-firebase-authentication-react-tutorial/), you will find a step by step guide on how to use Firebase authentication (sign up, sign in, sign out, …) in React. Beyond that you will use Firebase’s realtime database to store user entities. After that, it’s up to you to store more data of your application.
* **Code Organization:** On your way reading the book you came across one chapter about code organization. You could apply these changes now, if you haven’t done it yet. It will organize your components in structured files and folders (modules). In addition, it helps to understand and learn the principles of code splitting, reusability, maintainability and module API design.
* **Testing:** The book only scratched the surface of testing. If you are not familiar with the general topic, you could dive deeper into the concepts of unit testing and integration testing, especially in context of React applications. On an implementation level, I would recommend to stick to Enzyme and Jest in order to refine your approach of testing with unit tests and snapshot tests in React.
* **Asynchronous Requests:** You can substitute the native fetch API with third-party alterna-

tives to perform asynchronous requests: [superagent¹⁵²](https://github.com/visionmedia/superagent) or [axios¹⁵³](https://github.com/mzabriskie/axios). There is no perfect solution to make asynchronous requests. But by exchanging the buildings blocks around React, you make the experience [how powerful it can be to have this flexibility¹⁵⁴](https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/). In frameworks you usually stick to one solution. In a [flexible ecosystem like React¹⁵⁵](https://www.robinwieruch.de/essential-react-libraries-framework/) you can exchange the solutions.

* **Routing:** You can implement routing for your application with [react-router¹⁵⁶](https://github.com/ReactTraining/react-router). So far, you only have one page in your application. React Router helps you to have multiple pages across multiple URLs. When you introduce routing to your application, you don’t make any requests to your web server to fetch the next page. The router will do everything for you on the client-side.
* **Type Checking:** In one chapter, you have used React PropTypes to define component interfaces. It is a good practice to prevent bugs. But the PropTypes are only checked on runtime. You can go one step further to introduce static type checking on compile time. [TypeScript¹⁵⁷](https://www.typescriptlang.org/) is one popular approach. But in the React ecosystem, people often use [Flow¹⁵⁸](https://flowtype.org/). I can recommend to give Flow a shot if you are interested to make your application more robust.
* **Tooling with Webpack and Babel:** In the book you have used create-react-app to set up your application. At some point, when you have learned React, you might want to learn the

¹⁵¹<https://www.robinwieruch.de/complete-firebase-authentication-react-tutorial/>

¹⁵²<https://github.com/visionmedia/superagent>

¹⁵³<https://github.com/mzabriskie/axios>

¹⁵⁴<https://www.robinwieruch.de/reasons-why-i-moved-from-angular-to-react/>

¹⁵⁵<https://www.robinwieruch.de/essential-react-libraries-framework/>

¹⁵⁶<https://github.com/ReactTraining/react-router>

¹⁵⁷<https://www.typescriptlang.org/>

¹⁵⁸<https://flowtype.org/>

|  |  |
| --- | --- |
| Outline | 187 |

tooling around it. It enables you to setup your own project without create-react-app. I can recommend to follow a minimal setup with [Webpack and Babel¹⁵⁹](https://www.robinwieruch.de/minimal-react-webpack-babel-setup/). Afterward you can apply more tooling on your own. For instance, you could [use ESLint¹⁶⁰](https://www.robinwieruch.de/react-eslint-webpack-babel/) to follow a unified code style in your application.

* **React Native:** [React Native¹⁶¹](https://facebook.github.io/react-native/)brings your application on mobile devices. You can apply your learnings from React to ship iOS and Android applications. The learning curve, once you have learned React, shouldn’t be steep in React Native. Both share the same principles. You will only encounter different layout components on mobile than you are used to in web applications.

In general, I invite you to visit my [website¹⁶²](https://www.robinwieruch.de/) to find more interesting topics about web development and software engineering. You can [subscribe¹⁶³](https://www.getrevue.co/profile/rwieruch) to get updates roughly every month to your inbox and you can decide to support this kind of content by being my [Patron¹⁶⁴](https://www.patreon.com/rwieruch). Furthermore, the course platform [Road to React¹⁶⁵](https://roadtoreact.com/) offers more advanced courses to learn about the React ecosystem. You should check it out!

Once again, if you liked the book, I want you to take a moment to think about a person who would be a good match to learn React. Reach out to that person and share the book. It would mean a lot to me. The book is intended to be given to others. It will improve over time when more people read it and share their feedback with me.

Thank you a lot for reading the Road to learn React.

Robin

¹⁵⁹<https://www.robinwieruch.de/minimal-react-webpack-babel-setup/>

¹⁶⁰<https://www.robinwieruch.de/react-eslint-webpack-babel/>

¹⁶¹<https://facebook.github.io/react-native/>

¹⁶²<https://www.robinwieruch.de/>

¹⁶³<https://www.getrevue.co/profile/rwieruch>

¹⁶⁴<https://www.patreon.com/rwieruch>

¹⁶⁵<https://roadtoreact.com/>