

**U & I con reaccionan**

En la construcción de componentes U & I con Reaccionar

Farhad Ghayour

Este libro está a la venta en <http://leanpub.com/ui-react>

Esta versión se publicó el 08/10/2017



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.

© 2017 Farhad Ghayour

*Me gustaría agradecer al Dr. Kevin Healy, mi profesor de astronomía, que inspiró dentro de mí una sed de conocimiento que nunca será satisfecha. Me gustaría agradecer al Dr. Bernard Kobes y el Dr. John Devlin, mis profesores de filosofía, que me ha proporcionado herramientas, el rigor y la intuición de que he sido capaz de utilizar a través de todas mis pasiones e intereses. Me gustaría dar las gracias a las muchas personas maravillosas en Hack reactor y Famo.us que abrió las puertas de muchas oportunidades. También me gustaría dar las gracias a mis amigos cercanos y colegas que me han ayudado no sólo a ser un mejor profesional, sino también en una mejor persona.*

*Lo más importante, me gustaría agradecer a mi amante y el apoyo a la familia que siempre han estado ahí para mí.*

**Contenido**

|  |  |
| --- | --- |
| **Perfil** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **1** |
| Sobre el Autor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |
| Detalles de contacto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 1 |
| **Prefacio** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **2** |
| Temas. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| Capítulo 1 Introducción . . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| Capítulo 2: ¿Qué es T & I? . . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| Capítulo 3: Introducción. . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| Capítulo 4: La construcción de nuestra aplicación. . . . . . . . . . . . . . . . . . . . . . . | 3 |
| Capítulo 5: Uso de U & I Conceptos. . . . . . . . . . . . . . . . . . . . . | 3 |
| Capítulo 6: Explorando Preprocesadores CSS. . . . . . . . . . . . . . . . . | 3 |
| Capítulo 7: Exploración de módulos CSS. . . . . . . . . . . . . . . . . . . | 3 |
| Capítulo 8: Exploración de estilos en línea. . . . . . . . . . . . . . . . . . . . | 3 |
| Capítulo 9: La adición de capacidades de tiempo real. . . . . . . . . . . . . . . . | 3 |
| Capítulo 10: Presentando. . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
| Capítulo 11: Mirando hacia el futuro. . . . . . . . . . . . . . . . . . . . . . . | 4 |
| Requisitos. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| Audiencia. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
| Convenciones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| Ejemplo de código. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| Realimentación . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| Erratas. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 7 |
| La piratería. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| Contribuciones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| Preguntas. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |

|  |  |
| --- | --- |
| CONTENIDO |  |
| **Capítulo 1 Introducción** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **10** |
| Una breve historia del desarrollo web. . . . . . . . . . . . . . . . . . . . | 10 |
| Estilos en línea. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 10 |
| CSS interna. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| CSS externo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| Los clasificadores CSS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 13 |
| Un cambio necesario. . . . . . . . . . . . . . . . . . . . . . . . . . . | 15 |
| La subida de los componentes. . . . . . . . . . . . . . . . . . . . . . . . . . . | dieciséis |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | dieciséis |

|  |  |
| --- | --- |
| **Capítulo 2: ¿Qué es T & I?** . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **17** |
| Componentes de interfaz de usuario. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 17 |
| U & I Componentes. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 18 |
| Muestra T & I Contrato. . . . . . . . . . . . . . . . . . . . . . . . . . . | 19 |
| No hay espacio de nombres global. . . . . . . . . . . . . . . . . . . . . | 19 |
| estilos unidireccionales. . . . . . . . . . . . . . . . . . . . . | 20 |
| eliminación de código muerto. . . . . . . . . . . . . . . . . . . . | 21 |
| Minificación. . . . . . . . . . . . . . . . . . . . . . . . . . | 21 |
| constantes de compartir. . . . . . . . . . . . . . . . . . . . . . | 21 |
| Resolución determinista. . . . . . . . . . . . . . . . . . . | 21 |
| El aislamiento. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 21 |
| Extensible. . . . . . . . . . . . . . . . . . . . . . . . . . . | 21 |
| Documentable. . . . . . . . . . . . . . . . . . . . . . . . . | 21 |
| Presentable. . . . . . . . . . . . . . . . . . . . . . . . . . | 22 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 22 |

|  |  |
| --- | --- |
| **Capítulo 3: Introducción** . . . . . . . . . . . . . . . . . . . . . . . . . . . | **23** |
| Herramientas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 24 |
| Webpack. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 24 |
| JavaScript moderna con Babel. . . . . . . . . . . . . . . . . . . . . . | 25 |
| JSX. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 25 |
| ESLint. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 26 |
| Otros . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 26 |
| Nuestro proyecto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 27 |
| Repetitivo. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 27 |
| Estructura general . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 29 |

|  |  |
| --- | --- |
| CONTENIDO |  |
| Una pizarra limpia . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 31 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 32 |
| **Capítulo 4: La construcción de nuestra aplicación** . . . . . . . . . . . . . . . . . . . . . . . . . . | **33** |
| Primer pase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 33 |
| El estilo de nuestra aplicación. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 42 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 45 |
| **Capítulo 5: Uso de U & I Conceptos** . . . . . . . . . . . . . . . . . . . . . . . . | **46** |
| Organización. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 46 |
| Componentes necesarios. . . . . . . . . . . . . . . . . . . . . . . . . . . | 51 |
| En <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 52 |
| En <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 52 |
| En <TodosListItem />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 53 |
| ¿Otros? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 53 |
| Construcción <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . . . | 54 |
| Construcción <TodosListItem />. . . . . . . . . . . . . . . . . . . . . . . . | 57 |
| Considerando U & I Conceptos. . . . . . . . . . . . . . . . . . . . . . . . . | 62 |
| Nombre Espacio entre componentes. . . . . . . . . . . . . . . . . . . . . . . | 62 |
| Aplicación Nombre espaciado. . . . . . . . . . . . . . . . . . | 62 |
| Nombre del Componente espaciado. . . . . . . . . . . . . . . . . . | 62 |
| estilo unidireccional. . . . . . . . . . . . . . . . . . . . . . . . . . | 63 |
| Extensibilidad. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 64 |
| En acción . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 64 |
| La implementación de la aplicación Nombre espaciado. . . . . . . . . . . . . . | 64 |
| Ejecución Nombre del Componente espaciado. . . . . . . . . . | sesenta y cinco |
| La implementación extensibles y unidireccionales Estilos. . . . . | 71 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 73 |
| **Capítulo 6: Explorando Preprocesadores CSS** . . . . . . . . . . . . . . . . . . . | **75** |
| ¿Qué es un preprocesador de CSS? . . . . . . . . . . . . . . . . . . . . . . . . | 75 |
| ¿Por qué utilizar un preprocesador? . . . . . . . . . . . . . . . . . . . . . . . . . . | 75 |
| Conoce Sass. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 75 |
| Sass en acción. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 76 |
| Configuración de los estilos. . . . . . . . . . . . . . . . . . . . . . . . . . | 78 |
| Refactorización <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 81 |

|  |  |
| --- | --- |
| CONTENIDO |  |
| Refactorización <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . | 82 |
| Refactorización <TodosListItem />. . . . . . . . . . . . . . . . . . . . . . | 83 |
| Mejoras. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 84 |
| En <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 84 |
| Construcción <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . . . | 85 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 89 |
| **Capítulo 7: Exploración de módulos CSS** . . . . . . . . . . . . . . . . . . . . . . | **91** |
| ¿Cuáles son los módulos de CSS? . . . . . . . . . . . . . . . . . . . . . . . . . . . | 91 |
| ¿Por qué usar CSS módulos? . . . . . . . . . . . . . . . . . . . . . . . . . . . | 92 |
| Módulos CSS en acción. . . . . . . . . . . . . . . . . . . . . . . . . . . | 93 |
| refactoring estilos / \*. SCSS . . . . . . . . . . . . . . . . . . . . . . . | 94 |
| Refactorización <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 94 |
| Refactorización <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . | 97 |
| Refactorización <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . | 98 |
| Refactorización <TodosListItem />. . . . . . . . . . . . . . . . . . . . . . | 100 |
| Ejercicio sugerido. . . . . . . . . . . . . . . . . . . . . . | 103 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 103 |
| **Capítulo 8: Exploración de estilos en línea** . . . . . . . . . . . . . . . . . . . . . . . | **105** |
| ¿Cuáles son los estilos en línea. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 105 |
| Estilos en línea en acción. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 106 |
| Configuración de los estilos. . . . . . . . . . . . . . . . . . . . . . . . . . | 106 |
| Refactorización <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 107 |
| Refactorización <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . | 109 |
| Refactorización <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . | 111 |
| Refactorización <TodosListItems />. . . . . . . . . . . . . . . . . . . . . | 112 |
| Estilos en línea mejorada. . . . . . . . . . . . . . . . . . . . . . . . . . . . | 115 |
| ¿Cuál es el radio? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 116 |
| El radio de acción. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 116 |
| Configuración de los estilos. . . . . . . . . . . . . . . . . . . . . . . . . . | 116 |
| Refactorización <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 117 |
| Refactorización <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . | 120 |
| Refactorización <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . | 121 |
| Refactorización <TodosListItems />. . . . . . . . . . . . . . . . . . . . . | 121 |
| Ejercicio sugerido. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 124 |

|  |  |
| --- | --- |
| CONTENIDO |  |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 124 |
| **Capítulo 9: La adición de capacidades de tiempo real** . . . . . . . . . . . . . . . . . . | **126** |
| ¿Cuál es el tema Wrap? . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 126 |
| Wrap tema en acción. . . . . . . . . . . . . . . . . . . . . . . . . . . | 126 |
| Refactorización <app />. . . . . . . . . . . . . . . . . . . . . . . . . . . | 127 |
| Refactorización <TodosList />. . . . . . . . . . . . . . . . . . . . . . . . | 128 |
| Refactorización <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . . | 130 |
| Refactorización <TodosListItem />. . . . . . . . . . . . . . . . . . . . . . | 131 |
| Tema dinámico. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 134 |
| Mixins. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 140 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 145 |
| **Capítulo 10: Exposición** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | **147** |
| ¿Cuál es Storybook? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 147 |
| Libro de cuentos reaccionar en acción. . . . . . . . . . . . . . . . . . . . . . . . . . | 147 |
| Documentar <TodosList />. . . . . . . . . . . . . . . . . . . . . . . | 149 |
| Documentar <TodosListInfo />. . . . . . . . . . . . . . . . . . . . . | 158 |
| Documentar <TodosListItem />. . . . . . . . . . . . . . . . . . . . . | 159 |
| Resumen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 162 |
| **Capítulo 11: El anticipar** . . . . . . . . . . . . . . . . . . . . . . . . . . . | **164** |
| Exploraciones. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 164 |
| Siguiente CSS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 164 |
| CSS en JS. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 164 |
| Acelerado por hardware de interfaz de usuario. . . . . . . . . . . . . . . . . . . . . . . . | 165 |
| Conclusión. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 165 |

**Perfil**

**Autor:** Farhad Ghayour

**Sobre el Autor**

Farhad Ghayour es un consultor de tecnología con sede en San Francisco, CA, donde ayuda a transformar las compañías de Fortune 500 en todo el mundo. Anteriormente, fue un ingeniero de la plataforma en Famo.us la construcción de un motor de renderizado 3D WebGL y un ingeniero de software de plomo en varias nuevas empresas innovadoras de todo el mundo. Es un apasionado de toda la filosofía cosas, las matemáticas, el código y el diseño - y cada cierto tiempo se le puede encontrar en la escena de automóviles de alta potencia.

**Detalles de contacto**

* Email: [me@farhadg.com](mailto:me@farhadg.com)1
* Gorjeo: [@farhadg\_com](http://www.twitter.com/farhadg_com)2
* GitHub: [github.com/farhadg](http://www.github.com/farhadg)3
* LinkedIn: [linkedin.com/in/farhadg](https://www.linkedin.com/in/farhadg)4
* Sitio web: [farhadg.com](http://www.farhadg.com/)5

1[mailto: me@farhadg.com](mailto:me@farhadg.com)

* <http://www.twitter.com/farhadg_com>

3<http://www.github.com/farhadg>

4<https://www.linkedin.com/in/farhadg>

5[http://www.farhadg.com](http://www.farhadg.com/)

**Prefacio**

desarrollo frontend ha sufrido una gran transformación con tecnologías modernas frontend, como Reaccionar, angular, Vue, y así sucesivamente. Esto se debe en gran parte a la amplia adopción de la arquitectura basada en componentes que proporcionan estas tecnologías enorme éxito.

Este libro tiene como objetivo no sólo guiar a los lectores a partir de los bloques de construcción fundamentales de la creación de interfaces bien estructurados, sino también proporcionar una lista exhaustiva de las diferentes filosofías para crear modular, extensible y escalable T & I - es decir, la interfaz de usuario destinada a todos los desarrolladores, como usted y yo

**temas**

A pesar de que los conceptos de este libro se enseñan a través Reaccionar, que son transferibles a otras tecnologías frontend. Elegí Reaccionar como nuestra base debido a la simplicidad, la adopción de ancho, y poder innegable que proporciona.

**Capítulo 1 Introducción**

Explica una breve historia del desarrollo de la tela y el surgimiento de una arquitectura basada en componentes.

**Capítulo 2: ¿Qué es T & I?**

Proporciona una visión en profundidad de U & I y los criterios para la construcción de ellos.

**Capítulo 3: Introducción**

Esboza algunas de las tecnologías de base que usaremos y luego nos paseos por la creación de una base de código repetitivo.

|  |  |
| --- | --- |
| Prefacio | 3 |

**Capítulo 4: La construcción de nuestra aplicación**

nos guía a través de la construcción de la base de nuestra aplicación.

**Capítulo 5: Uso de U & I Conceptos**

nos guía en nuestra aplicación refactorización utilizando las mejores prácticas U & I.

**Capítulo 6: Explorando Preprocesadores CSS**

Explica preprocesadores CSS y nos guía en nuestra interfaz refactorización utilizando Sass.

**Capítulo 7: Exploración de módulos CSS**

Presenta módulos CSS y nos guía en nuestra interfaz refactorización utilizando módulos CSS.

**Capítulo 8: Exploración de estilos en línea**

nos lleva en un viaje a través de los estilos en línea y nos guía en la refactorización nuestra aplicación utilizando esta filosofía de fecha en un entorno moderno.

**Capítulo 9: La adición de capacidades de tiempo real**

Presenta una perspectiva completamente nueva de ver las interfaces interactivas y ayuda a llevar nuestra aplicación a la vida con capacidades de tiempo real.

**Capítulo 10: Exposición**

nos guía sobre cómo podemos construir, documentar y mostrar nuestros componentes en un entorno aislado.

|  |  |
| --- | --- |
| Prefacio | 4 |

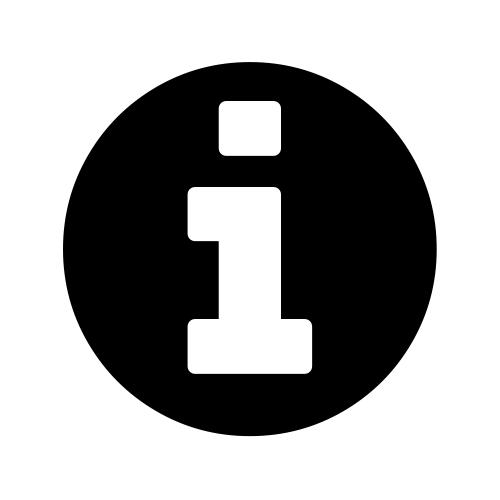
**Capítulo 11: El anticipar**

Compara las diferentes metodologías cubiertos en este libro, con una visión hacia el futuro del desarrollo de la interfaz de usuario.

**requisitos**

Se recomienda el siguiente para una máxima productividad:

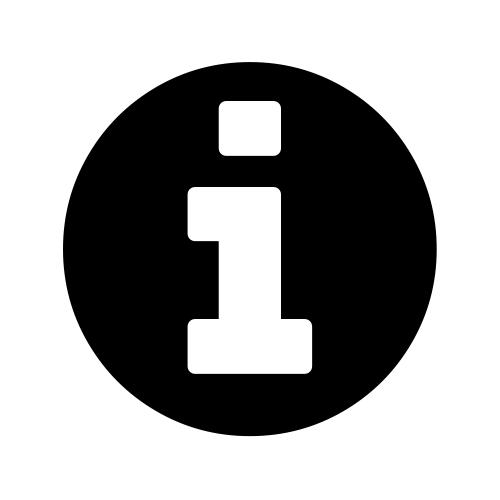
* Cualquier ordenador moderno con Linux, Mac OS o Windows.
* Nodo
* NGP
* Git (opcional)



Todo el software mencionado en este libro son gratis y se puede descargar desde Internet. Puede encontrar el código fuente alojado en GitHub:<https://github.com/FarhadG/ui-react>6.

**Audiencia**

Este libro es ideal para los desarrolladores que están familiarizados con reaccionar y están en busca de una guía completa para la construcción de interfaces de usuario modular, extensible y escalable. A pesar de que este libro está destinado para el intermedio a avanzado Reaccionar desarrolladores, cualquier persona nueva a reaccionar fácilmente pueden seguir a lo largo dado el formato progresivo de este libro, donde cada capítulo se basa en los capítulos antes.



Las ideas se tratan en este libro no están atados a reaccionar y son transferibles a otras tecnologías frontend.

* <https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Prefacio | 5 |

**convenciones**

En este libro, se encuentra un número de formatos de texto que transmiten diferentes tipos de información. He aquí algunos ejemplos para la ilustración:

Un bloque de código:

* función greet (nombre) {
* retorno 'Hola,' + nombre;

3 }

Cuando los bloques de código son redundantes de los ejemplos anteriores, verá una elipsis que significa que se han omitido para ser capaz de llamar su atención sobre los bloques apropiados de código:

* función greet (nombre) {
* ...

3 }

Para significar bloques adicionales de código, que serán resaltados en negrita:

* función greet (nombre) {
* ...

3 }

4

5 **saludar ( 'desarrollador estrella de rock!');**

Para ayudar a localizar los archivos de los que estamos trabajando, verá títulos en la parte superior de los bloques de código que describen la ruta al archivo:

|  |  |
| --- | --- |
| Prefacio | 6 |

**src / util / greet.js**

* función greet (nombre) {
* ...

3 }

4

5 ...

A menudo es útil para proporcionar comentarios en línea dentro de los ejemplos de código, como se puede ver aquí:

**src / util / greet.js**

* función greet (nombre) {
* // si el nombre no existe, que está establecido en un valor predeterminado

3name = nombre || 'Mundo';

* ...

5 }

Cualquier entrada o salida de línea de comandos se escribe como sigue:

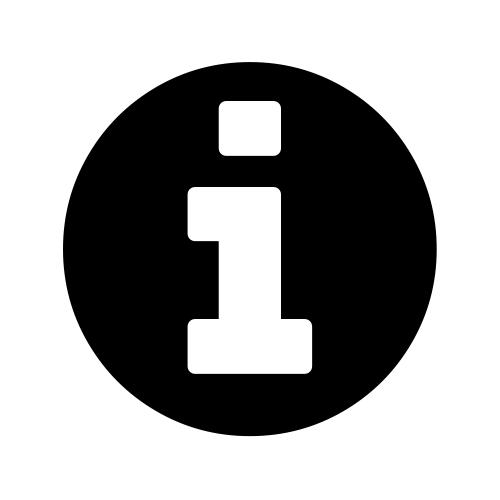
$ NPM instalación --save reaccionan

$ NPM instalar --save-dev webpack

$ NPM instalar --save-dev estilo-loader

los pssignifica una nueva entrada de línea de comandos. Esto no tiene por qué ser incluidos con sus comandos ya que sólo sirve para distinguir las nuevas entradas de línea de comandos.

Por último, cualquier información o consejos útiles auxiliares se pueden encontrar en el siguiente formato:



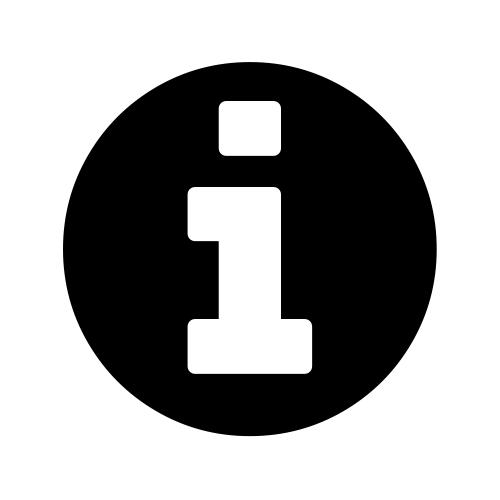
Webpack tiene muchos cargadores potentes y útiles que figuran en su página web para referencia.

|  |  |
| --- | --- |
| Prefacio | 7 |

**ejemplo de código**

Puede descargar el código de ejemplo para este libro por ir a la GitHub Reposi-toria, <https://github.com/FarhadG/ui-react>7. Hay muchas opciones para descargar el código fuente:

* **GIT:** Puede clonar el repositorio e ir a través de los distintos capítulos de la navegación de los diferentes directorios.
* **zip descarga**: Se puede descargar directamente el código base de GitHub a través de su opción de postal.



El contenido de este libro se pueden encontrar bajo la manuscrito directorio y el código fuente se ha desglosado por cada capítulo bajo el códigodirectorio. Si desea ejecutar el código fuente, puede hacerlo mediante la instalación de dependencias del capítulo y ejecutando el apropiadoNPM guiones.

**Realimentación**

Su regeneración es muy importante para mí, así que por favor no dude en extender la mano y me dejó saber lo que te gustó de este libro, ¿cuáles son sus deficiencias y cómo puedo mejorar sobre ellos. Puede encontrar más detalles de contacto en el**perfil** sección.

**Errata**

A pesar de que he tomado todas las precauciones para asegurar la exactitud de los contenidos de este libro, los errores ocurren. Si encuentra cualquier error en el contenido del libro, por favor, póngase en contacto conmigo o utilizar el[cuestiones de GitHub](https://github.com/FarhadG/ui-react)8página para informar del problema. Al informar estos errores, se le ayudará a mejorar la calidad del libro para usted y otros lectores en el futuro. Una vez que su fe de erratas se verifican, su presentación será aceptada y se resolverá la fe de erratas. Puede encontrar más detalles de contacto en el**perfil** sección.

* <https://github.com/FarhadG/ui-react>

8<https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Prefacio | 8 |

**Piratería**

Aunque estoy muy a favor de código abierto, la piratería de material con copyright es un problema importante. Si se encuentra con todas las copias ilegales de este libro, por favor, póngase en contacto conmigo. Puede encontrar más detalles de contacto en el**perfil** sección.

**Aportes**

Dado que todos los contenidos en este libro son de código abierto y provisto intencionalmente en GitHub, las contribuciones son bienvenidos y alentados. Puede encontrar toda la información necesaria sobre[GitHub](https://github.com/FarhadG/ui-react)9.

**preguntas**

Si tienes alguna pregunta, duda o sugerencia, puede ponerse en contacto conmigo directamente. Puede encontrar más detalles de contacto en el**perfil** sección.

* <https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Prefacio | 9 |

**¿Qué estamos esperando? Empecemos…**

**Capítulo 1 Introducción**

A pesar de que estamos a punto de embarcarse en un nuevo viaje juntos, este recorrido se extiende desde hace muchos años atrás. Vamos a tomar un momento rápido y repasar un poco de historia que abarca la evolución del desarrollo de interfaz de usuario.

**Una breve historia del desarrollo Web**

Recuerdo los días en que comenzó el desarrollo de interfaz de usuario mediante interfaces de primeros de diseño en Photoshop y luego “extraídos” usando HTML, CSS y JavaScript mínima. El proceso de construcción de un sitio web implicaría el uso de HTML para el marcado, el JavaScript para un comportamiento muy simple y CSS para la presentación. A pesar de que seguimos utilizando las mismas tecnologías para la entrega de una funcionalidad similar, las metodologías han cambiado con el tiempo. Vamos a echar un simple punto de vista de las diversas metodologías que han llegado a buen término a través de la lente de la CSS.

**estilos en línea**

Puede utilizar el atributo de estilo en la etiqueta correspondiente. El atributo de estilo puede contener cualquier propiedad CSS.

**index.html**

* <! DOCTYPE html>

2 <Html>

* <Body>
* <Div style = "background-color: negro">
* <Style p = "color: rojo; font-size: 18px;">

6¡Hola Mundo!

* </ P>
* </ Div>

9</ Body>

1. </ Html>

|  |  |
| --- | --- |
| Capítulo 1 Introducción | 11 |

**CSS interna**

Dado que el contenido estilos en línea combinación con la presentación, muchos estilos se duplicarían través de los elementos. Aquí es donde las hojas de estilo CSS vienen al rescate.

**index.html**

* <! DOCTYPE html>

2 <Html>

* <Head>

4

5<! - de estilo CSS interna ->

6<Style type = "text / css">

* div {
* background-color: negro;
* }

10

1. pag {
2. color rojo;
3. font-size: 18px;
4. }
5. </ Style>

dieciséis

1. </ Head>
2. <Body>
3. <Div>
4. <P> Hola, mundo! </ P>
5. </ Div>
6. </ Body>
7. </ Html>

**CSS externo**

Como sitios web crecen en complejidad y comienzan a contener varias páginas, podemos extraer styleheets CSS en archivos separados. Una convención común es a base separada o global

|  |  |
| --- | --- |
| Capítulo 1 Introducción | 12 |

estilos que abarcan páginas en una sola hoja de estilo de las hojas de estilo en la página individuales. Esto tiene la ventaja de ser capaz de mantener nuestro código “seco” de manera que podemos aprovechar los mismos estilos a través de múltiples páginas y fácilmente cambiar los estilos de elementos similares en toda nuestra página web.

**styles.css**

* div {
* background-color: negro;

3 }

4

5 pag {

6color rojo;

7font-size: 18px;

8 }

**index.html**

* <! DOCTYPE html>

2 <Html>

* <Head>
* <! - Referencias Nuestros estilos de archivos ->
* <Link rel = "stylesheet" type = "text / css" href = "./ style.css">
* </ Head>

7

* <Body>
* <Div>

1. <P> Hola, mundo! </ P>
2. </ Div>
3. </ Body>
4. </ Html>

|  |  |
| --- | --- |
| Capítulo 1 Introducción | 13 |

**about.html**

* <! DOCTYPE html>

2 <Html>

* <Head>
* <! - Referencias Nuestros estilos de archivos ->
* <Link rel = "stylesheet" type = "text / css" href = "./ style.css">
* </ Head>

7

* <Body>
* <Div>

1. <P> Sobre el mundo. </ P>
2. <P> Para más información básica acerca de nuestro increíble sitio web. </ P>
3. </ Div>
4. </ Body>
5. </ Html>

**Los clasificadores CSS**

La extracción de estos estilos funciona bien, pero no podemos escalar nuestra aplicación, ya que todos los estilos aplicados son globales para todos los elementos HTML coincidentes. Por lo tanto, utilizamos clases CSS y los identificadores de ser capaz de dirigirse a los elementos según sea necesario.

**styles.css**

* .envase {
* background-color: negro;

3 }

4

5 .Message {

6color rojo;

7font-size: 18px;

8 }

9

1. .info {
2. color amarillo;

|  |  |
| --- | --- |
| Capítulo 1 Introducción | 14 |

1. font-size: 13px;
2. }

**index.html**

* <! DOCTYPE html>

2 <Html>

* <Head>
* <Link rel = "stylesheet" type = "text / css" href = "./ style.css">
* </ Head>

6

* <Body>
* <Div class = "contenedor">
* <P class = "mensaje"> Hola, mundo! </ P>

1. </ Div>
2. </ Body>
3. </ Html>

**about.html**

* <! DOCTYPE html>

2 <Html>

* <Head>
* <Link rel = "stylesheet" type = "text / css" href = "./ style.css">
* </ Head>

6

* <Body>
* <Div class = "contenedor">
* <P class = "mensaje"> Sobre el mundo. </ P>

1. <P class = "info"> Parte de la información útil sobre nuestros sorprendentes webs \
2. ite. </ p>
3. </ Div>
4. </ Body>
5. </ Html>

|  |  |
| --- | --- |
| Capítulo 1 Introducción | 15 |

Sorprendentemente, la aplicación y la organización de nuestra JavaScript pasó por una historia muy similar. Como puede haber adivinado, nosotros:

1. ellos inline
2. ellos se extrae a la parte superior de nuestro documento usando un interno <Script> etiqueta
3. organizados en su propia separada Js archivos, y, en última instancia
4. referenciados elementos individuales por sus clasificadores.

**Un cambio necesario**

La evolución de las prácticas de codificación ha sido grande y las abstracciones tenían sentido por su tiempo, sin embargo, la mayor adopción de aplicaciones de una sola página ha empezado a sacudir nuestras ideas fundamentales del desarrollo de interfaz de usuario. Algunos de los escollos con nuestras ideas anteriores incluyen:

* **Todo es global.** Los selectores se comparan con todo en el DOM, por lo que requieren estrategias de asignación de nombres inteligentes que son difíciles de aplicar y fácil de romper, para combatir las colisiones.
* **La creciente complejidad es aterradora.** Ita € ™ s no inusual para que los desarrolladores reconocen que tienen miedo a modificar su propio CSS y JavaScript. Dada la complejidad aterradora de su código, que preferirían agregar a la base de código, que arriesgarse a romper la funcionalidad mediante la refactorización o eliminación de código.
* **La gestión de las interacciones de estado y de usuario a través de elementos de la interfaz es un desastre.** Aunque no tenemos bibliotecas JS simples que nos permiten administrar y conectar diferentes componentes de la interfaz, las interacciones entre los componentes de frecuencia para hacer una base de código desordenado.

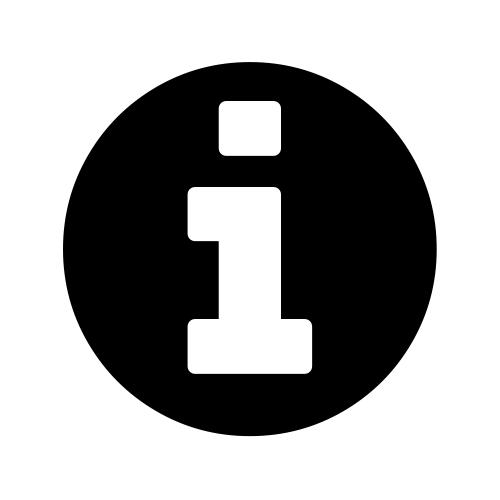
aplicaciones de una sola página han exigido nuestra atención para repensar nuevos paradigmas para la escritura de los componentes encapsulados y modulares. Es decir, elementos discretos, funcional y encapsulados de interfaz de usuario. Aquellos de nosotros que trató a CSS y JavaScript, como ciudadanos de primera clase están empezando a darse cuenta de la potencia de los componentes de interfaz de usuario como los bloques de construcción de nuestras aplicaciones. Ahora apuntamos a componentes modulares que se pueden compartir a través de plataformas y aplicaciones, teniendo en cuenta el tipo correcto de las limitaciones y entornos.

|  |  |
| --- | --- |
| Capítulo 1 Introducción | dieciséis |

**El ascenso de Componentes**

Reaccionar fue una de las primeras bibliotecas de interfaz de usuario que me hizo pensar en una arquitectura basada en componentes. diseño basado en componentes es de gran alcance para los diseñadores y desarrolladores. Se ayuda a los diseñadores en la construcción de bellos elementos de la interfaz modulares que pueden ser diseñados, probados y revisados ​​de forma aislada y ayuda a los desarrolladores en la reflexión sobre la función más importante de cada elemento de la interfaz.

Reaccionar no sólo promueve muchos grandes patrones pero también empuja a los desarrolladores de frontend para pensar en la interfaz de usuario de una manera modular y escalable. Como vamos a venir a ver en los capítulos siguientes, Reaccionar se presta bien a los componentes modulares. Dicho esto, el estilo ha sido y continúa siendo un problema importante en la construcción de estos componentes modulares. Vamos a cubrir muchas mejores prácticas para la construcción de componentes que abarque diversas estrategias eficaces.



Muchos de los conceptos de este libro están cubiertos a través de la lente de estilos, pero están intrínsecamente ligados a la arquitectura basada en componentes. Es decir, planificar, organizar y construir nuestro margen de beneficio, el comportamiento y estilos en alineación con la especificación de componentes.

**Resumen**

En este capítulo, hemos cubierto una historia rápida de desarrollo de interfaz de usuario y establecer las bases para lo que está por venir con una breve introducción a los conceptos de componentes modulares.

Prepárese para un viaje emocionante! Estamos a punto de entrar en componentes U & I y los planos para la construcción de ellos en el capítulo siguiente.

**Capítulo 2: ¿Qué es T & I?**

En el capítulo anterior hemos cubierto una breve historia del desarrollo de interfaz de usuario y se introdujeron a algunos términos clave que que trataremos en este libro. En este capítulo, vamos a cubrir lo que los componentes U & I son y cómo derivar de ellos de una manera axiomática.

**Por lo tanto, lo que es U & I?**

En resumen, si definimos nuestros componentes de interfaz de usuario en términos de componentes modulares que son reutilizables en una aplicación dada, a continuación, los componentes U & I pueden ser entendidas como componentes de interfaz de usuario que puede ser utilizado a través de aplicaciones.



componentes U & I son componentes de interfaz de usuario que son aplicaciones a través reutilizables.

Una de mis principales motivaciones para pensar acerca de los componentes U & I fue una [hablar](https://vimeo.com/116209150)10 dada por [Christopher Chedeau](https://twitter.com/Vjeux)11, Uno de Facebook del propio, en noviembre de 2014 a NationJS. La charla cubre muchas de las caídas de CSS en el desarrollo de interfaz de usuario y la forma en que el equipo de Facebook ha estado explorando estilos en línea para mitigar estos problemas. A pesar de que la charla sólo se habla de CSS, he encontrado muchos de los conceptos a ser grandes axiomas para definir los componentes U & I. Ahora que usted tiene una sugerencia para qué componentes U & I son, vamos a cubrir en detalle y cómo ir sobre la construcción de ellos.

**componentes de interfaz de usuario**

UI componentes son bloques de interfaz de usuario modulares que pueden ser utilizados a través de una aplicación. Su funcionalidad y capacidad de reutilización difieren de una aplicación a otra, pero su intención general, desde mi experiencia, ha sido ámbito para una sola organización o aplicación.

1. <https://vimeo.com/116209150>
2. <https://twitter.com/Vjeux>

|  |  |
| --- | --- |
| Capítulo 2: ¿Qué es T & I? | 18 |

Por ejemplo, una aplicación puede tener una Encabezamiento, Pie de página y deslizador componente que se utiliza en toda la aplicación, sin embargo, el alcance, la modularidad, extensibilidad y escalabilidad de estos componentes siempre se ha entendido en un solo contexto - es decir, una sola aplicación.

**U & I Componentes**

componentes U & I son componentes de interfaz de usuario que están destinados a varios contextos. Es decir, que son componentes destinados no sólo para una sola aplicación, pero varios-aplica ciones. Esto puede sonar simple, pero la verdad es que la construcción de componentes UI reutilizables en una sola aplicación es lo suficientemente difícil, por lo que la construcción de componentes U & I para varias aplicaciones requiere aún más rigor. Una preocupación válida es si existe una diferencia entre una biblioteca de U & I y algunas de las bibliotecas de interfaz tradicionales que hemos visto en los últimos años. Hay una diferencia sutil que quedará más claro a medida que definimos una muestra U & I contraigan. En resumen, todas las bibliotecas U & I son las bibliotecas de interfaz de usuario, pero no todas las bibliotecas de interfaz de usuario son bibliotecas U & I.



Todas las bibliotecas U & I son las bibliotecas de interfaz de usuario, pero no todas las bibliotecas de interfaz de usuario son bibliotecas U & I.

Otra preocupación es válida si realmente necesitamos considerar la construcción de tales componentes-izada generales. Argumento que debemos, no porque necesariamente tienen componentes generalizadas en todos los casos, sino que debemos construir sobre bases mantener y escalable.

Uno de los puntos de partida clave para la construcción de componentes U & I es un contrato bien definido. Los contratos se definen criterios de cuidado para los editores y los consumidores de referencia de componente. Por ejemplo, si definimos un contrato de U & I que incluyeestilos unidireccionales, Entonces muchas de las bibliotecas de interfaz tradicionales no pasaría nuestra U & I spec.

Vamos a empezar con una muestra de contrato de U & I para una aplicación que vamos a construir a lo largo de este libro. Me gustaría dejar claro que estos requisitos no son ni una lista exhaustiva ni los criterios para la construcción de componentes U & I. Simplemente sirve como una

|  |  |
| --- | --- |
| Capítulo 2: ¿Qué es T & I? | 19 |

lista de requisitos que estamos definiendo para permitir que para nosotros construir y consumir una biblioteca U & I.

**Muestra T & I Contrato**

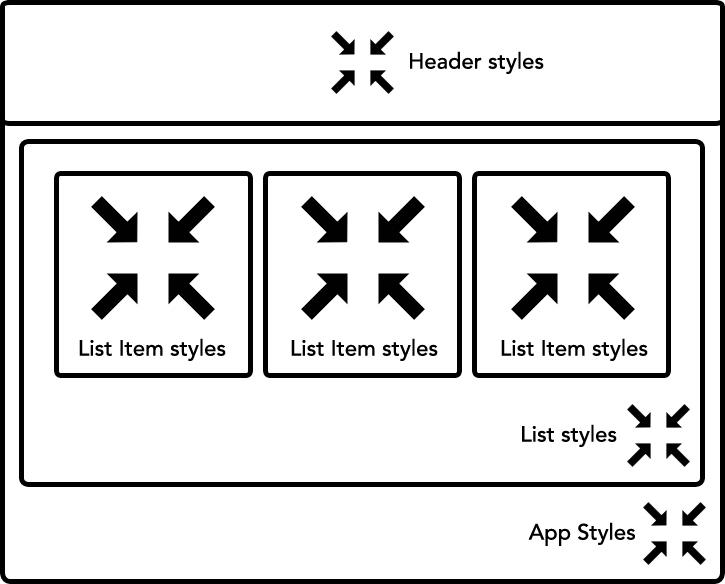
Aquí está una lista exhaustiva de los requisitos que trataremos en este libro. Algunos de estos requisitos será ignorado o dejado sólo sin control a cumplirse en capítulos posteriores cuando sea apropiado temporalmente.

**No hay espacio de nombres global**

Desde selectores CSS son globales, es necesario evitar cualquier posibilidad de nombrar colisión.

|  |  |
| --- | --- |
| Capítulo 2: ¿Qué es T & I? | 20 |

**estilos unidireccionales**



**estilos unidireccionales**

Dado que los componentes se pueden utilizar en todas las aplicaciones, es necesario para los estilos de componentes a ser unidireccional; es decir, los componentes hijos no pueden modificar estilos fuera de su alcance.



Como regla general, el diseño centrado atributos CSS como margen, anchura, altura, Etc. y CSS selectores que extenderse más allá del alcance de su componente debe ser evitado.

|  |  |
| --- | --- |
| Capítulo 2: ¿Qué es T & I? | 21 |

**eliminación de código muerto**

Como queremos que inclinarse y código mantenible, es necesario poder eliminar mediante programación código innecesario.

**minificación**

El tamaño del código es importante. Por lo tanto, es necesario ser capaz de minify nuestro código, incluyendo nuestros estilos.

**Las constantes se pueden compartir**

las interfaces modernas son altamente interactivos. Por lo tanto, es necesario ser capaz de compartir constantes entre HTML, CSS y JavaScript.

**resolución determinista**

comportamiento de los componentes debe ser predecible. Por lo tanto, es necesario evitar situaciones donde los estilos de carga de forma asíncrona puede dar lugar a resultados diferentes.

**Aislamiento**

Dado que los componentes deben ser modular, es necesario que sus estilos y comportamientos a ser aislados y bien encapsulados.

**extensible**

Dado que los componentes se pueden utilizar en muchos contextos, es necesario ser capaz de extender sus estilos y comportamiento.

**documentable**

Dado que los componentes se pueden utilizar en muchos contextos, es necesario para que su interfaz es documentable.

|  |  |
| --- | --- |
| Capítulo 2: ¿Qué es T & I? | 22 |

**Presentable**

Dado que los componentes se pueden utilizar en muchos contextos, es necesaria para que sean presentables en forma aislada.

**Resumen**

En este capítulo, exploramos qué componentes U & I son y cómo conceptualmente difieren de sus contrapartes tradicionales. También se define una muestra U & I contrato que nos guiará a lo largo de este libro.

No hay ningún problema si algunos de estos conceptos son ajenas a ti, porque vamos a continuar referencia a ellos en un entorno más práctico a medida que comenzamos la codificación en los siguientes capítulos.

**Capítulo 3: Introducción**

En el capítulo anterior hemos cubierto una visión general conceptual de los componentes U & I. Estos conceptos U & I no son útiles a menos que nos basamos nuestro conocimiento con la práctica. Este será el primer capítulo de una serie de implementaciones que abarcan varias estrategias para la construcción de nuestra aplicación. Debido a las diferencias drásticas entre estas estrategias U & I, que será útil para experimentar de primera mano sus pros y contras.

En este capítulo, vamos a construir un texto modelo simple de conseguir nosotros hacia arriba y corriendo. Boilerplates son tedioso y engorroso, especialmente en el mundo de JavaScript; el número de herramientas y tecnologías para elegir puede ser desalentador. Además, Reaccionar del sistema modular y no prescriptiva eco no hace esto más fácil. Así, en lugar de tratar con configuraciones de construcción complejas y de trabajo hasta un dolor de cabeza, vamos a confiar en Facebook de práctica[Create-reaccionar aplicación](https://github.com/facebookincubator/create-react-app)12utilidad de línea de comandos. Si no está familiarizado con este módulo,Create-reaccionar aplicaciónes un módulo de línea de comandos muy práctico creado por Cara-libro que resume algunos de los mejores y más recientes herramientas de la comunidad de reaccionar durante una experiencia de desarrollo agradable. Por ejemplo, vamos a tener tecnologías como webpack para liar, Babel como nuestro transpiler ES6, el módulo de recarga caliente para el desarrollo más rápido, ESLint para mantener nuestro código limpio y consistente, y muchos otros que están configurados fuera de la caja.

Si no lo dices, yo: Eso es increíble!

Su [documentación](https://github.com/facebookincubator/create-react-app)13es exhaustiva y útil. Le recomiendo leerlo, si va a ir más allá de los conceptos básicos tratados en este libro.



Asegúrese de tener nodo y las otras tecnologías mencionadas en el prefacio.

Si tienes algún problema, puede hacer referencia a código fuente de cada capítulo sobre [GitHub](https://github.com/FarhadG/ui-react)14.

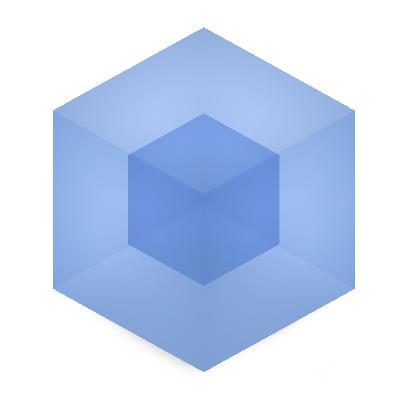
1. <https://github.com/facebookincubator/create-react-app>
2. <https://github.com/facebookincubator/create-react-app>
3. <https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 24 |

**Herramientas**

Es posible que ya esté familiarizado con algunas de las herramientas de Create-reaccionar aplicación, Pero debemos cubrir los fundamentos para asegurar que tenemos un punto de partida similar.

**webpack**



**webpack**

[webpack](https://webpack.github.io/)15 es un potente bundler módulo. UN *haz* es un archivo JavaScript que incor-tasas activos que pertenecen juntos y deben ser servidos al cliente en una respuesta a una solicitud de una sola fila. Un paquete puede incluir JavaScript, estilos CSS, HTML, y casi cualquier otro tipo de archivo.

Webpack vaga sobre el código fuente de la aplicación, en busca de importardeclaraciones, la construcción de un gráfico de la dependencia, y que emiten uno o más paquetes. Con plugins y reglas, webpack puede preprocesar y minify diferentes archivos que no son de JavaScript como mecanografiado, Sass, menos, y muchos otros.

1. <https://webpack.github.io/>

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 25 |

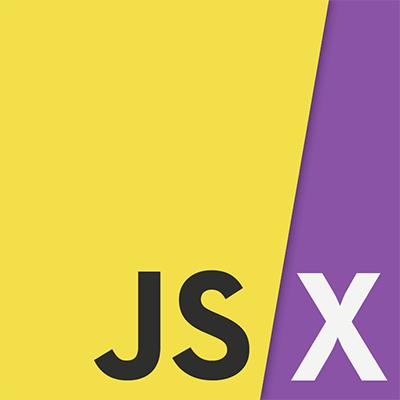
**JavaScript moderna con Babel**



**Babel**

[Babel](http://babeljs.io/)dieciséis es un transpiler JavaScript, mejor conocido por su capacidad para convertir ES6 y más allá (las próximas versiones de JavaScript) en código que se ejecuta en el navegador (o en el servidor) en la actualidad.

**JSX**



**JSX**

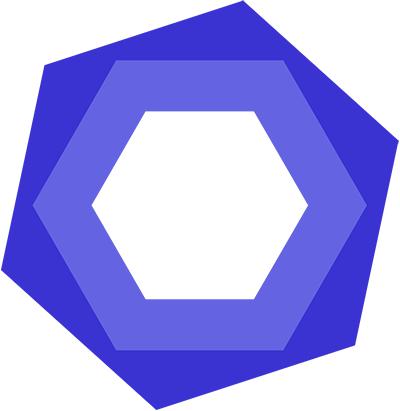
[JSX](https://jsx.github.io/)17 es un paso preprocesador que se suma a la sintaxis XML JavaScript. Al igual que XML, JSX

1. <http://babeljs.io/>
2. <https://jsx.github.io/>

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 26 |

etiquetas tienen una etiqueta con su nombre, atributos y niños. Usted pueden utilizar definitivamente reaccionar sin JSX pero JSX hace Reaccionar mucho más elegante.

**ESLint**



**ESLint**

[ESLint](http://eslint.org/)18 es una fuente abierta JavaScript linting utilidad. Código [linting](http://en.wikipedia.org/wiki/Lint_(software))19 es un tipo de análisis estático que se utiliza con frecuencia para encontrar patrones problemáticos o código que no se adhieren a ciertas normas de estilo. Con ESLint, los desarrolladores pueden escribir reglas para configurar un estándar de codificación para un proyecto determinado.

**Otros**

Como se mencionó anteriormente, hay un sinnúmero de herramientas que Create-reaccionar aplicaciónconfigura para nosotros. Si usted está interesado en aprender más, usted debe explorar su[repositorio](https://github.com/facebookincubator/create-react-app)20.

1. <http://eslint.org/>
2. <http://en.wikipedia.org/wiki/Lint_(software)>
3. <https://github.com/facebookincubator/create-react-app>

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 27 |

**Nuestro proyecto**

¿Estás listo? ¡Empecemos! Pero, espere ... lo que estamos construyendo? Vamos a construir una manera muy sencilla aplicación que muestra una lista de todos. Vamos a mantener la aplicación simple, por lo que nos centramos en la arquitectura y las mejores prácticas como base para la interfaz de usuario escalable.

Dicho esto, me recomienda tomar los componentes básicos de cada capítulo para construir una funcionalidad más compleja en la aplicación. Esto asegurará que los conceptos presentados en este libro se refuerzan con la práctica creativa y de detección.

**repetitivo**

Vamos a instalar a nivel mundial Create-reaccionar aplicación por lo que podemos usarlo desde la línea de comandos:

$ NPM instalar -g crear una reacción aplicación

Una vez instalado, podemos ejecutar el siguiente comando:

$ Crear una reacción aplicación ui-reaccionar

Si todo se ha realizado correctamente, nuestra ui-reaccionardirectorio debe ser generada y todas las dependencias necesarias se instaló. También puede ver una lista de útilesguiones después de la instalación:

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 28 |

inicio del hilo

# Inicia el servidor de desarrollo.

acumulación de hilo

# Haces de la aplicación en archivos estáticos para la producción.

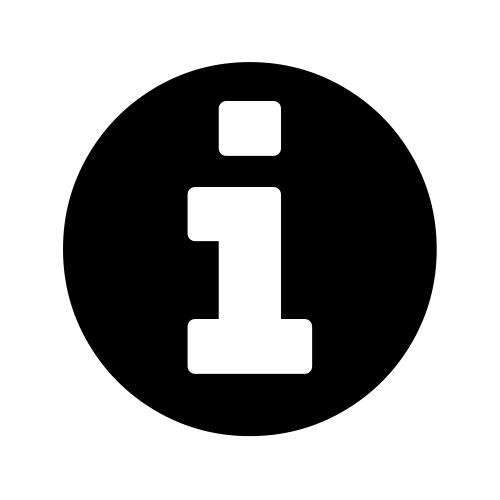
prueba de hilo

# Inicia el corredor de prueba.

expulsión del hilo

* Elimina esta herramienta y copias construir dependencias,
* archivos de configuración y scripts en la aplicación
* directorio. Si hace esto, no se puede volver!

Para todos los efectos, vamos a tratar hilo y NPMcomo siendo la misma. Si usted está interesado en aprender más acerca de estos gestores de paquetes, hay un montón de recursos que explican sus diferencias. Dicho esto, se puede ejecutar comandos con cualquieraNPM plazo [comando] o su hilo equivalente.



Vamos a utilizar la mayor parte de estas secuencias de comandos, a excepción de prueba ya que hay un sinnúmero de recursos en línea que hacen un gran trabajo explicando cómo poner a prueba su Reaccionar aplicación.

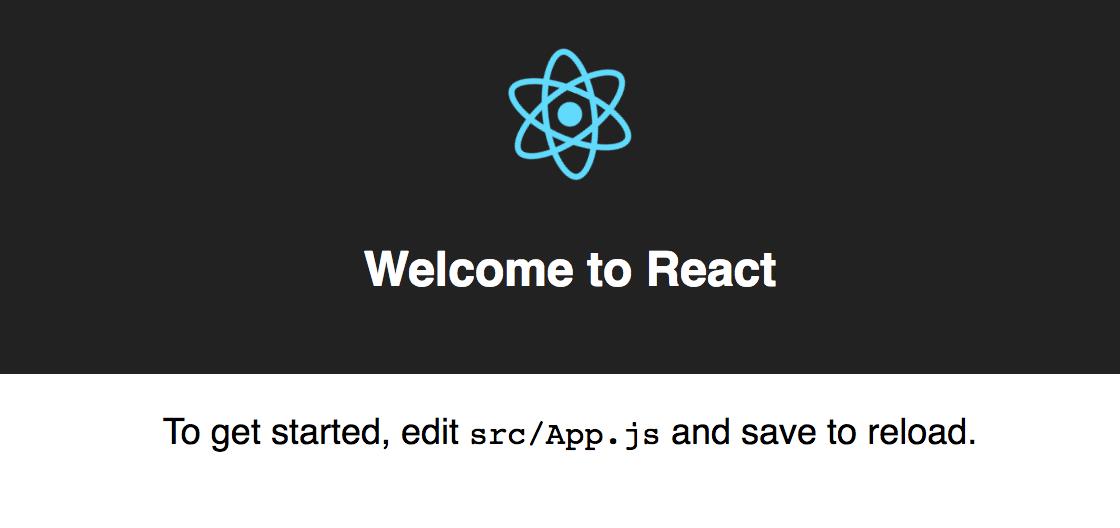
Vamos a ejecutar los siguientes comandos y empezar nuestra aplicación:

* cd-ui reaccionar
* inicio NPM

Usted debe ser dirigido a [http: // localhost: 3000 /](http://localhost:3000/)21 para ver la aplicación en ejecución.

1. [http: // localhost: 3000 /](http://localhost:3000/)

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 29 |



Es bastante notable que todas las herramientas que van desde un conjunto completo de pruebas, pelusa, recarga caliente, transpiling, etc., han sido atendidos por nosotros con un solo comando. Esto supone un considerable ahorro de tiempo, de modo que podemos centrarnos en las partes importantes de por qué usted está leyendo este libro.

**Estructura general**

Vaya por delante y abrir la aplicación en el editor de su elección para que podamos empezar a construir nuestra aplicación. Vamos a empezar por la construcción de la versión genérica que vamos a utilizar como un medio para ir a través de las diversas estrategias de U & I cubiertos este libro.

En nuestro directorio de la aplicación, veremos los siguientes directorios y archivos:

.gitignore

package.json

público/

favicon.ico

index.html

README.md

src /

App.css

app.js

App.test.js

index.css

index.js

logo.svg

Vamos a pasar la mayor parte de nuestro tiempo en el interior de la srcdirectorio. Una gran característica queCreate-reaccionar aplicación proporciona es una *centrado en código* semilla ocultando el código repetitivo,

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 30 |

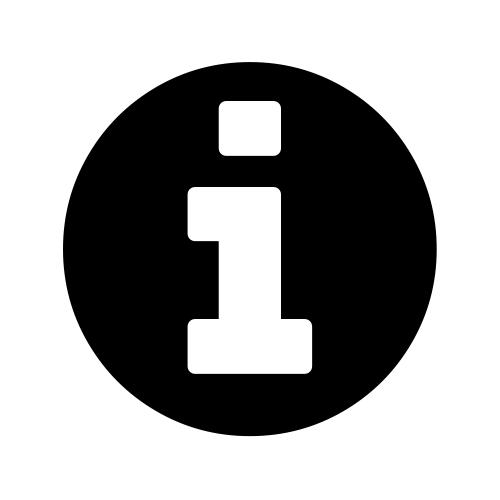
configuraciones y secuencias de comandos. Dicho esto, tendremos que configurarwebpack a medida que avanzamos a través de los capítulos, por lo que vamos a expulsar ahora y tener un control total desde el inicio de nuestro proceso de desarrollo.



La opción de expulsar desde el principio del proyecto no es obligatorio, ya que es útil para disminuir la carga cognitiva de navegar a una semilla más grande a menos que sea necesario.

Vamos expulsar desde el Create-reaccionar aplicación semilla administrado:

$ NPM carrera de expulsión



Este comando es permanente e irreversible.

Después de ejecutar el comando, usted debe ver que algunos directorios y archivos adicionales fueron inyectados en el proyecto de la semilla.

.gitignore

config / configuraciones # bundler

package.json

público/

favicon.ico

index.html

README.md

scripts / guiones # nodo

src /

App.css

app.js

App.test.js

index.css

index.js

logo.svg

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 31 |

**Una pizarra limpia**

Vamos a empezar con una pizarra limpia, así que adelante y quitar todo, desde la src directorio y crear un nuevo archivo llamado unos pocos index.js:

**src / index.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* {} la importación de render 'reaccionar-dom';

4

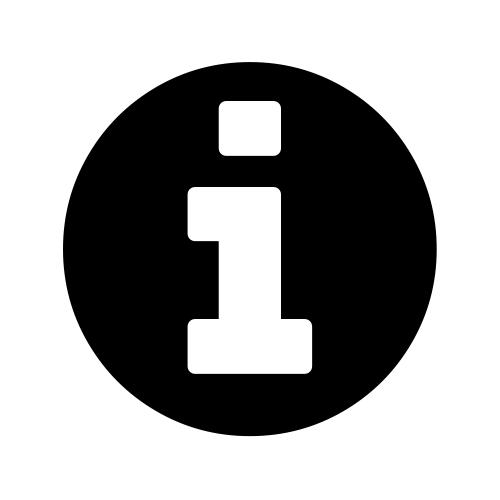
5 const App = () => (

* <Div>
* <H1> ¡Hola, mundo! </ H1>
* </ Div>

9 );

10

1. hacer(
2. <App />,
3. document.getElementById ( 'root')
4. );



En el momento de escribir este libro, Reaccionar componentes deben devolver un solo nodo Reaccionar, por lo que tenemos que garantizar siempre nuestros componentes están envueltos en un único elemento HTML. Esto puede cambiar en el futuro cercano.

Vamos a utilizar algunas de las técnicas más avanzadas desde el código JavaScript y reaccionar comu-nidad, según sea necesario, para construir nuestra aplicación. Como se puede ver arriba, estamos usando ES6 deconstruir nuestras dependencias y estamos utilizando la sintaxis de componente puro de React para definirAplicación.

Si no está familiarizado con ES6, le recomiendo estos [recursos gratuitos](https://github.com/getify/You-Dont-Know-JS)22 para convertirse en un ninja de ES6.



1. <https://github.com/getify/You-Dont-Know-JS>

|  |  |
| --- | --- |
| Capítulo 3: Introducción | 32 |

Si usted visita su navegador, debería ver nuestro amable ¡Hola Mundo! mensaje.



**Resumen**

En este capítulo, hemos cubierto algunas de las tecnologías que se utilizarán en los capítulos siguientes, además de Facebook de Create-reaccionar aplicación utilidad de línea de comandos realizar el arranque rápidamente nuestra aplicación.

En el siguiente capítulo, vamos a empezar la construcción de la primera versión de nuestra aplicación utilizando una estrategia más tradicional basada en componentes.

**Capítulo 4: La construcción de nuestra aplicación**

En el capítulo anterior hemos utilizado Create-reaccionar aplicación para configurar rápidamente un proyecto de semillas.

En este capítulo, vamos a iniciar la construcción de la primera versión de nuestra aplicación.

Vamos a empezar por crear todo dentro de index.jsy más tarde refactorizar todo en componentes separados. Dicho esto, por lo general es más fácil construir com-ponentes de manera aislada, pero teniendo en cuenta que estamos empezando, que será más fácil de ver todo en un solo archivo para que podamos capturar todo el viaje juntos.

**Primer pase**

primero vamos a instalar [Lodash](https://lodash.com/)23“Una moderna biblioteca de utilidades JavaScript entrega modularidad, el rendimiento y los extras” con el siguiente comando:

$ NPM instalar --save lodash

Una vez instalado, podemos comenzar con código y cubrirlo con detalle después:

**src / index.js**

* // dependencias
* \_ importar de 'lodash';
* Reaccionar importación, {} Componente de 'reaccionar';

4 {} la importación de render 'reaccionar-dom';

5

6 clase App extiende Componente {

7

8Constructor (... args) {

9super (... args);

1. <https://lodash.com/>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 34 |

1. this.state = {
2. Todos: {}
3. };
4. }

14

1. componentDidMount () {
2. this.setState ({
3. Todos: {
4. 1: {id: 1, completado: falso, descripción: 'tarea 1'},
5. 2: {id: 2, completado: falso, descripción: 'la tarea 2'},
6. 3: {id: 3, completado: falso, descripción: 'tarea 3'},
7. 4: {id: 4, completado: falso, descripción: 'la tarea 4'}
8. }
9. });
10. }

25

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <Ul>
5. {\_.map (todos, (TODO, id) =>
6. <Li clave = {id}>

32 {} Todo.description

1. </ Li>
2. )}
3. </ Ul>
4. );
5. }
6. }

39

1. hacer(
2. <App />,
3. document.getElementById ( 'root')
4. );

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 35 |

Hemos importado lodashpara ayudarnos con las tareas de programación comunes de una manera elegante y concisa. Hemos convertido nuestraAplicación componente de un componente puro Reaccionar a su claseinterfaz, ya que necesitamos el estado del componente interno para nuestro todos. En una aplicación real, que tendría que utilizar un gestor de estado, tales como[Redux](http://redux.js.org/)24, [Flujo](https://facebook.github.io/flux/)25, [MOBX](https://mobx.js.org/)26, Etc, pero no vamos a necesitar una aquí.

Para minimizar las dependencias externas, no vamos a golpear a una API a buscar a nuestros datos, pero vamos a tratar el establecimiento de nuestro estado como si fuéramos. Dentro decomponentDidMount, Se puede ver que estamos actualizando de nuestro estado todoscon una lista de todos. La estructura de datos para un elemento de tarea es la siguiente:

1: {

id: 1,

completado: falso,

Descripción: 'tarea 1'

}

* carné de identidad: Un identificador único asociado a cada TODO
* terminado: Un booleano para determinar el estado del TODO
* descripción: La descripción de la TODO

¿Por qué almacenar los todos como una objeto en lugar de dentro de una formación? Opté por ellos han almacenado en unaobjeto con cada TODO carné de identidad como la clave, ya que es más fácil hacer referencia a un individuo de tareas y podemos hacerlo en un tiempo constante.

En nuestro hacer método, estamos agarrando y cartografía sobre nuestra colección de todos a través de Lodash \_.mapa y las hacen tan li elementos dentro de una ul.

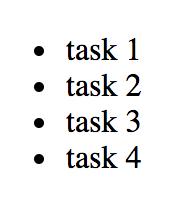


Es una buena práctica utilizar el llave pilar derecho por elementos que estamos bucle terminado, por lo que reaccionan de forma inteligente puede realizar un seguimiento de ellos y hacer optimizaciones internas.

Por el momento, vamos a seguir estos simples sin mucho estilo. Si usted visita su navegador, debería ver una lista de todos 4 en una representaciónul etiqueta.

1. <http://redux.js.org/>
2. <https://facebook.github.io/flux/>
3. <https://mobx.js.org/>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 36 |



Ahora que tenemos nuestros todos, ¿cómo camiar su estado? Vamos a configurar manualmente uno de nuestros todos para completar y luego definir algunos estilos para un TODO completado:

**src / index.js**

...

componentDidMount () {

this.setState ({

Todos: {

1. {Id: 1, completado: falso, descripción: 'tarea 1'}, // establecer manualmente a la verdadera
2. {Id: 2, completado: true, descripción: 'la tarea 2'},
3. {Id: 3, completado: falso, descripción: 'tarea 3'},
4. {Id: 4, completado: falso, descripción: 'la tarea 4'}

}

});

}

...

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 37 |

Agrega un terminado la clase a la li elementos que son completa:

**src / index.js**

...

<Ul>

{\_.map (todos, (TODO, id) =>

<Li clave = {id}

className = {todo.completed? 'Completado': ''}> {} todo.description

</ Li>

)}

</ Ul>

...

Vamos a crear una styles.css presentar en virtud del src directorio con un conjunto básico de estilos:

**src / styles.css**

* cuerpo {
* fondo: negro;

3color blanco;

4 }

5

6 .completed {

7color rojo;

8text-decoration: tachado;

9 }

Vamos a aprovechar webpack e importar nuestra css dentro index.js:

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 38 |

**src / index.js**

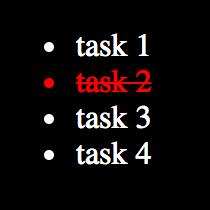
* dependencias

...

* dependencias locales de importación'./styles.css

...

Visita su navegador y verá los nuevos estilos aplicados.



Ya que no modificable nuestra que hacerRealiza en su totalidad estado, ¿cómo podemos dar al usuario la habilidad de ser capaz de cambiar dinámicamente ese estado?

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 39 |

**src / index.js**

...

toggleTodo (id, e) {

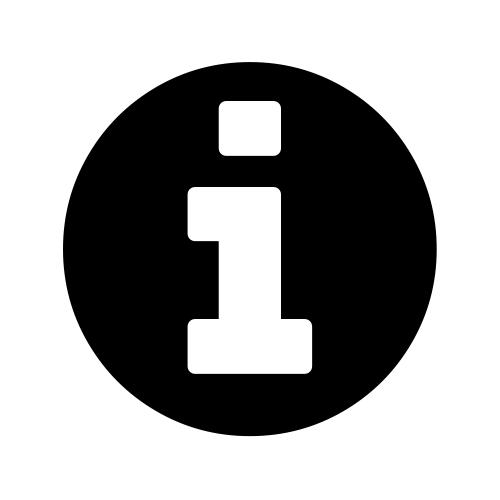
e.preventDefault ();

todos const = \_.clone (this.state.todos); todos [id] = .completed todos [id] .completed!; this.setState ({todos});

}

...

Añadimos el siguiente método, toggleTodo, para nuestro Aplicaciónclase, que nos da la capacidad de cambiar nuestra TODO. En lugar de mutar el estado original de nuestra todos, estamos clonación de una nueva colección de modo que no nos encontramos con situaciones en las que las mutaciones provocan un comportamiento impredecible. Si este es un tema ajeno a usted, le recomiendo leer más sobre[inmutabilidad](https://en.wikipedia.org/wiki/Immutable_object)27 dentro de los paradigmas de programación funcional orientada a objetos y.



Si usted está interesado en las estructuras de datos inmutables de JavaScript, [Estoy-mutableJS](https://facebook.github.io/immutable-js/)28 es un gran candidato de código abierto.

Tenemos que hacer nuestros todos desencadenan nuestro nuevo toggleTodo método:

1. <https://en.wikipedia.org/wiki/Immutable_object>
2. <https://facebook.github.io/immutable-js/>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 40 |

**src / index.js**

...

render () {

const {todos} = this.state;

regreso (

<Ul>

{\_.map (todos, (TODO, id) =>

<Li clave = {id}

className = {todo.completed? 'Completado': ''} onclick = {(E) => this.toggleTodo (id, e)}>

{} Todo.description

</ Li>

)}

</ Ul>

);

}

...

Si usted ha estado siguiendo a lo largo de, su index.js debería tener este aspecto:

**src / index.js**

* // dependencias
* \_ importar de 'lodash';
* Reaccionar importación, {} Componente de 'reaccionar';

4 {} la importación de render 'reaccionar-dom';

5

6 // dependencias locales

7 importación' ./styles.css'

8

9 clase App extiende Componente {

10

1. Constructor (... args) {

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 41 |

1. super (... args);
2. this.state = {
3. Todos: {}
4. };
5. }

17

1. componentDidMount () {
2. this.setState ({
3. Todos: {
4. 1: {id: 1, completado: falso, descripción: 'tarea 1'},
5. // define como true
6. 2: {id: 2, completado: true, descripción: 'la tarea 2'},
7. 3: {id: 3, completado: falso, descripción: 'tarea 3'},
8. 4: {id: 4, completado: falso, descripción: 'la tarea 4'}
9. }
10. });
11. }

29

1. toggleTodo (id, e) {
2. e.preventDefault ();
3. todos const = \_.clone (this.state.todos);
4. todos [id] = .completed todos [id] .completed!;
5. this.setState ({todos});
6. }

36

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <Ul>
5. {\_.map (todos, (TODO, id) =>
6. <Li clave = {id}

|  |  |
| --- | --- |
| 43 | className = {todo.completed? 'Completado': ''} |
| 44 | onclick = {(E) => this.toggleTodo (id, e)}> |
| 45 | {} Todo.description |

1. </ Li>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 42 |

1. )}
2. </ Ul>
3. );
4. }
5. }

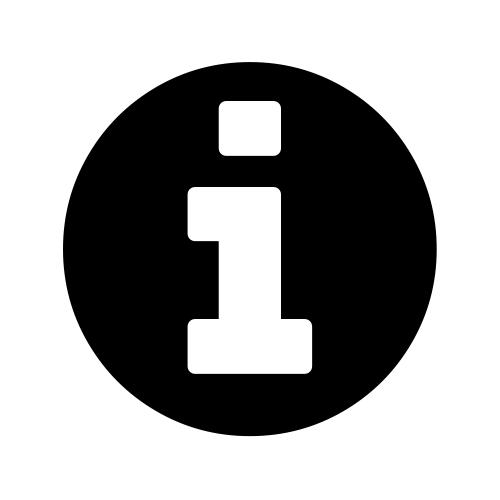
52

1. hacer(
2. <App />,
3. document.getElementById ( 'root')
4. );

¡Viola! Echa un vistazo a su navegador y tareas de distancia. Todos ellos marca como que no hay mañana!

**El estilo de nuestra aplicación**

¡Sé que sé! Nuestra aplicación se ve horrible. Voy a dejar el trabajo creativo de hacer nuestra aplicación totalmente fabuloso depende de usted.



Si desea compartir sus versiones beautified, tirar de solicitudes a la [código-base](https://github.com/FarhadG/ui-react)29 son más que bienvenidos.

Sin embargo, haremos nuestra aplicación vea algo decente con algunos estilos básicos para que nuestros ojos no nos odian por el final de este libro.

1. <https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 43 |

**src / styles.css**

* html {
* caja de tamaño: Frontera de la caja;

3 }

4

5 \*, \*:antes después {

6cuadro de dimensionamiento: heredar;

7 }

8

9 cuerpo {

1. background: # F1F1F1;
2. font-size: 10px;
3. font-family: "Helvetica Neue", "Arial", "sans-serif";
4. margin: 0;
5. padding: 0;
6. }

dieciséis

1. ul {
2. list-style: none;
3. margen: 50px auto;
4. max-width: 800px;
5. padding: 10px 15px;
6. }

23

1. li {
2. Antecedentes: #fafafa;
3. border-radius: 5px;
4. frontera: 1px solid # E1E1E1;
5. box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);
6. color: # 888888;
7. cursor: puntero;
8. font-size: 2rem;
9. margen: 10px 0;
10. padding: 15px 20px;
11. position: relative;

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 44 |

1. transición: todo 0.2s facilidad;
2. }

37

1. li: hover {
2. opacidad: 0,8;
3. }

41

1. .completed {
2. background: # E4E4E4;
3. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
4. Color: #AAAAAA;
5. text-decoration: tachado;
6. top: 3px;
7. }

Estos estilos son lo suficientemente delgado para proporcionar cierta elegancia y son lo suficientemente simple para ser compatibles entre navegadores.

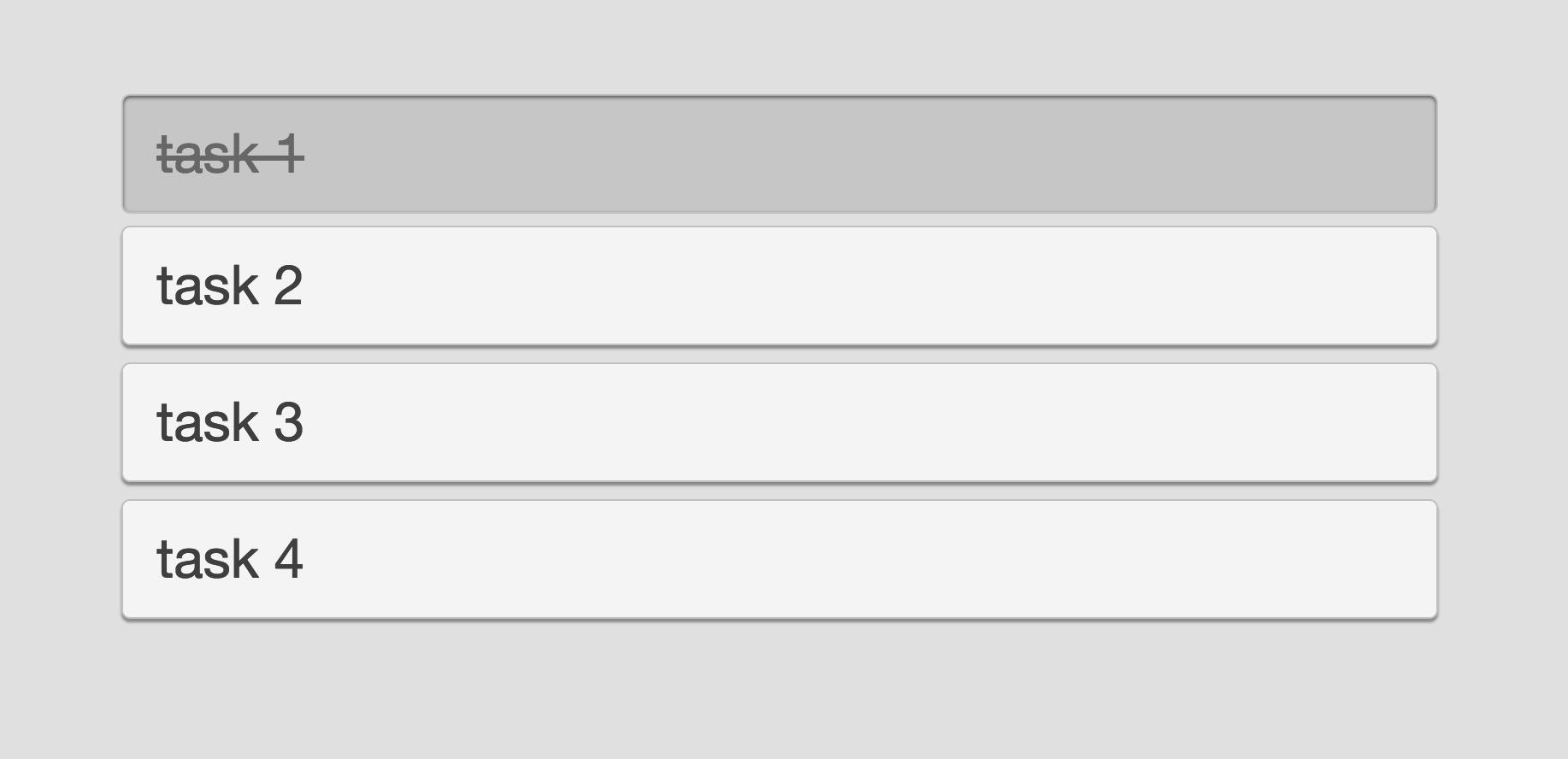
Construir herramientas, como webpack de [Autoprefixer](https://github.com/postcss/autoprefixer)30, Ayuda a mantener el CSS entre navegadores compatibles.



Si usted visita su navegador, debería ver nuestra aplicación recién refinado:

1. <https://github.com/postcss/autoprefixer>

|  |  |
| --- | --- |
| Capítulo 4: La construcción de nuestra aplicación | 45 |



**Resumen**

En este capítulo, se presentó a la base de nuestra aplicación. Hemos introducido algunos conceptos básicos que están bien establecidos en el ecosistema reaccionar y quedaron fuera otros que vamos a cubrir en los próximos capítulos.

Usted puede preguntarse cuando modularidad y encapsulación entran en juego? No se preocupe! Vamos a recibir en eso en el capítulo siguiente.

**Capítulo 5: Uso de U & I Conceptos**

En el capítulo anterior hemos construido el primer paso de nuestra aplicación utilizando las bases Ubicamos usando Create-reaccionar aplicación.

En este capítulo, vamos a refactorizar nuestra aplicación en componentes y cubrimos algunos conceptos U & I en ese proceso.

**Organización**

No podemos construir grandes aplicaciones dentro de un solo archivo, por lo que no vamos a hacer eso.

Crear un componentes directorio para albergar la totalidad de nuestros futuros componentes:

src /

componentes /

index.js

Vamos a crear nuestro primer componente:

src /

componentes /

app /

App.css

app.js

App.spec.js

index.js

Vamos a reunir todas las Aplicación código relacionado con el interior de la Aplicacióndirectorio. Como se puede ver, tenemos unacss, js y spec.js presentar la descripción de nuestra Aplicacióncomponente. Es convencional para ubicar todo el código relacionado componentes juntos, de manera que podamos escalar y organizar

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 47 |

nuestro código de una manera centrada en el componente. El enfoque de este libro no está poniendo a prueba, por lo que no va a trabajar con elspec.jsarchivos, pero quiero dar un ejemplo de cómo le gustaría organizar archivos relacionados. Cuando la organización de sus componentes, mantener la modularidad y encapsulación en cuenta; usted podría incluso ir tan lejos como poner los activos relevantes, tales como imágenes, iconos, etc. dentro de ese directorio.

Extraer Aplicación de index.js e incluirlo dentro de app.js:

**componentes / App / app.js**

* // dependencias
* \_ importar de 'lodash';
* Reaccionar importación, {} Componente de 'reaccionar';

4

5 // dependencias locales

6 importación' ./App.css';

7

8 clase predeterminada de exportación de aplicaciones se extiende Componente {

9

1. Constructor (... args) {
2. super (... args);
3. this.state = {
4. Todos: {}
5. };
6. }

dieciséis

1. componentDidMount () {
2. this.setState ({
3. Todos: {
4. 1: {id: 1, completado: falso, descripción: 'tarea 1'},
5. // define como true
6. 2: {id: 2, completado: true, descripción: 'la tarea 2'},
7. 3: {id: 3, completado: falso, descripción: 'tarea 3'},
8. 4: {id: 4, completado: falso, descripción: 'la tarea 4'}
9. }
10. });
11. }

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 48 |

28

1. toggleTodo (id, e) {
2. e.preventDefault ();
3. todos const = \_.clone (this.state.todos);
4. todos [id] = .completed todos [id] .completed!;
5. this.setState ({todos});
6. }

35

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <Ul>
5. {\_.map (todos, (TODO, id) =>
6. <Li clave = {id}

|  |  |
| --- | --- |
| 42 | className = {todo.completed? 'Completado': ''} |
| 43 | onclick = {(E) => this.toggleTodo (id, e)}> |
| 44 | {} Todo.description |

1. </ Li>
2. )}
3. </ Ul>
4. );
5. }
6. }

Tenga en cuenta la importada App.css el cual se contiene toda Aplicación estilos relacionados, como se ve a continuación:

**componentes / App / App.css**

* ul {
* list-style: none;

3margen: 50px auto;

4max-width: 800px;

5padding: 10px 15px;

6 }

7

8 li {

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 49 |

* Antecedentes: #fafafa;

10border-radius: 5px;

11frontera: 1px solid # E1E1E1;

12box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

13color: # 888888;

14cursor: puntero;

15font-size: 2rem;

dieciséismargen: 10px 0;

17padding: 15px 20px;

18position: relative;

19transición: todo 0.2s facilidad;

20 }

21

22 li: hover {

23opacidad: 0,8;

24 }

25

26 .completed {

27background: # E4E4E4;

28box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);

29Color: #AAAAAA;

30text-decoration: tachado;

31top: 3px;

32 }

Contiene casi todo, desde nuestra anterior styles.css presentar, sin los estilos globales que no están vinculados a Aplicación. Por último, vamos a la actualizaciónindex.js y styles.css para reflejar estos cambios:

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 50 |

**src / index.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* {} la importación de render 'reaccionar-dom';

4

5 // dependencias locales

6 Aplicación de importación './components/App/App';

7 importación' ./styles.css';

8

9 hacer(

1. <App />,
2. document.getElementById ( 'root')
3. );

Importamos nuestra Aplicacióncomponente y hacer todo igual que antes. Vamos a eliminar los duplicados de estilosstyles.css:

**src / styles.css**

* html {
* caja de tamaño: Frontera de la caja;

3 }

4

5 \*, \*:antes después {

6cuadro de dimensionamiento: heredar;

7 }

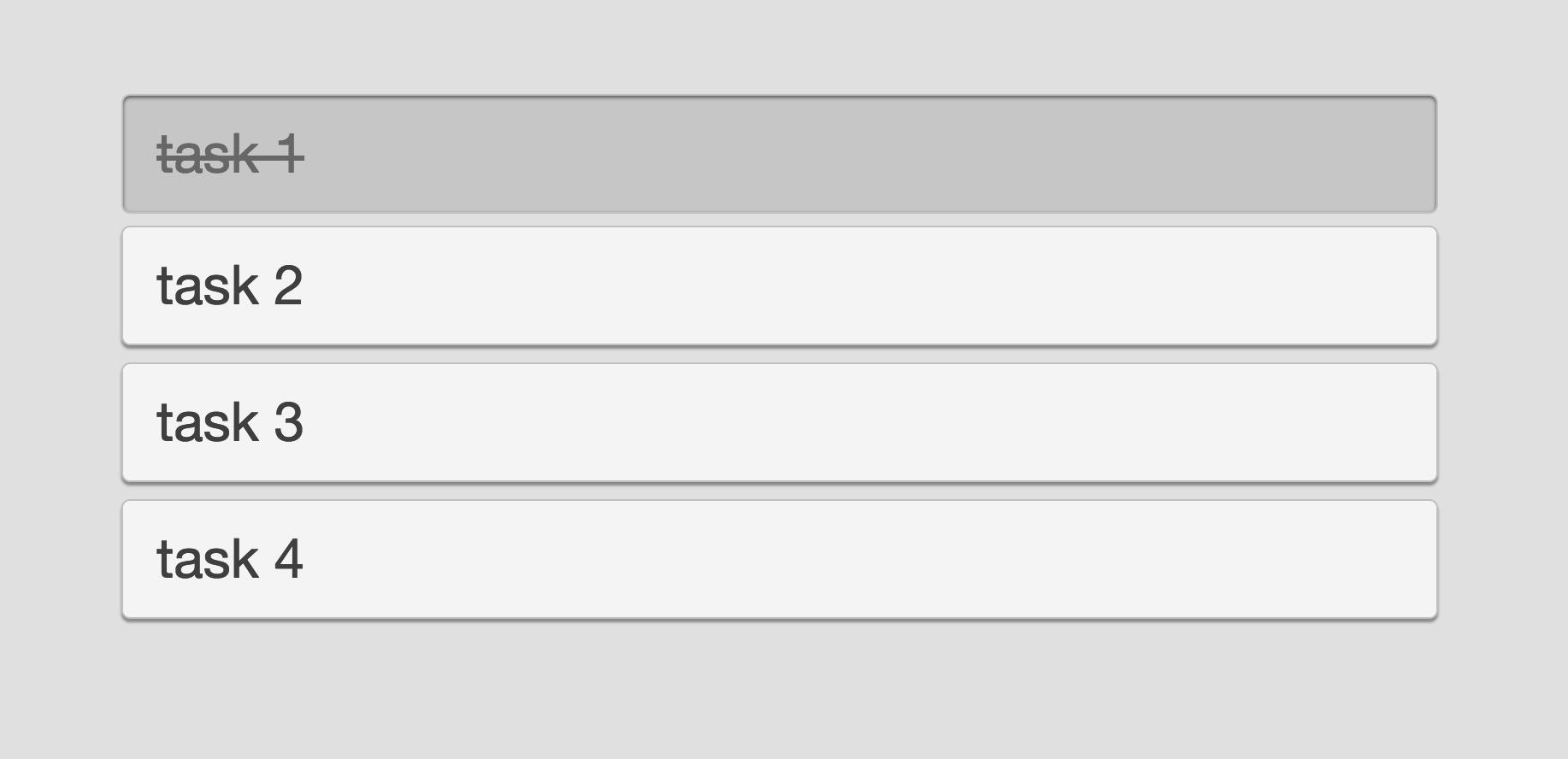
8

9 cuerpo {

1. background: # F1F1F1;
2. font-size: 10px;
3. font-family: "Helvetica Neue", "Arial", "sans-serif";
4. margin: 0;
5. padding: 0;
6. }

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 51 |

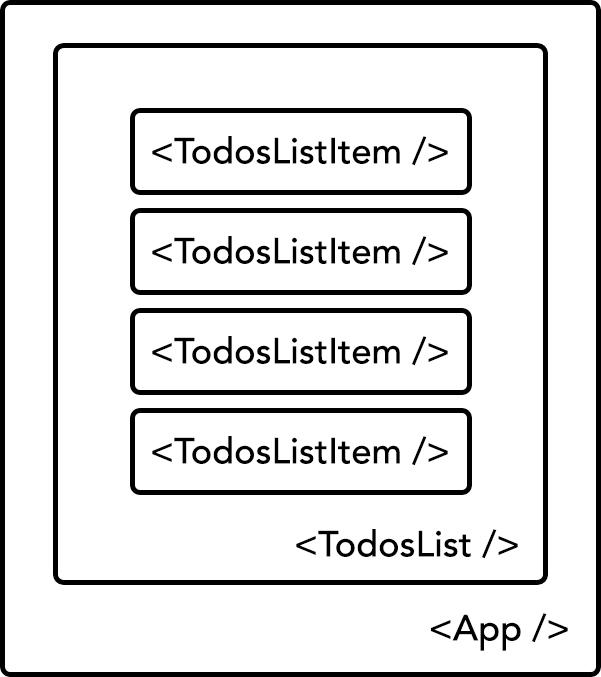
Si todo ha ido según lo previsto, debería ver nuestra aplicación render tal como lo hizo antes.



**componentes necesarios**

Eso es todo lo agradable y elegante, pero no hemos hecho mucho con este refactor, aparte de co-localizar nuestra Aplicacióncódigo relacionado. Antes de pasar a los bits más interesantes de nuestro refactorización, necesitamos delinear los componentes necesarios para construir nuestra aplicación. Este tipo de planificación es crucial para cualquier desarrollo U & I.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 52 |



**Diagrama de componentes**

**En <app />**

Nuestra Aplicacióncomponente será punto de entrada para nuestra aplicación, crear instancias de los componentes de nivel superior. En esencia, nos gustaría queAplicación hacer lo siguiente:

* ha podido recuperar un conjunto de todos
* gestionar éstos de todos
* actualizar Todos estos
* pasar la habilidad para actualizar Todos estos componentes a los niños
* hacer todos

**En <TodosList />**

Ya que Aplicación no se refiere a la real directamente todos, vamos a crear una TodosList componente, que servirá la siguiente propósito (s):

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 53 |

* aceptar una lista de todos
* hacer una lista de elementos de tareas

Tu puedes pensar en TodosList como una ordenado o desordenada lista que simplemente hace que cualquier conjunto de niñostransmitido a la misma. Usted puede preguntarse por qué tenemos que modularizar nuestraTodosList de esta manera, pero más adelante veremos el poder de separar los elementos de tareas reales de la lista de todos donde nuestro TodosList puede representar cualquier tipo de artículos, no sólo nuestros artículos de la lista de tareas.

**En <TodosListItem />**

Nuestra aplicación crea una instancia y gestiona todos nuestros. Tenemos una lista, que muestra una lista de los niños, pero no hemos definido el elemento de tarea real. Aquí es dondeTodosListItementra en la imagen. En nuestro paso anterior, he mencionado eso que es una buena idea SEP-arate nuestros elementos de tareas de la lista de tareas. A pesar de que no vamos a construir diferentes tipos de tareas pendientes elementos, puede, por ejemplo, imaginar casosTodosList puede hacer una variedad de diferentes elementos de tareas, tales como TodosListItem, TodosListItemWithIcon,

TodosListItemThatNeverGetsFinishedCuzItsRequiresWayTooMuchWorkToComplete, Etcétera. Estos son, por supuesto, ejemplos, pero es una buena idea tener estos separados, en lugar de tenerTodosList con destino a la prestación de un solo tipo de componente de elemento de tarea.

El punto de nuestra TodosListItem es bastante simple:

* aceptar a Todo
* aceptar una devolución de llamada para poder palanca el estado del TODO
* hacer el elemento de tarea

**¿Otros?**

Cualquier otro componente necesario? ¡No! ¡Eso es! Bien, al menos por ahora.

Como se mencionó anteriormente, vamos a mantener la aplicación mínima de modo que podemos centrarnos en las estrategias en lugar de la slue de otros temas que pueden sangrar en este dominio.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 54 |

**Construcción <TodosList />**

Pero primero, vamos a terminar lo que empezamos y refactorizar nuestros componentes para satisfacer las especificaciones de diseño se definió anteriormente. Vamos a crear unaTodosList directorio junto con sus archivos asociados.

src /

componentes /

app /

TodosList /

TodosList.css

TodosList.js

TodosList.spec.js

index.js

A continuación, cree TodosList como un ul que simplemente hace que la niños transmitido a través accesorios:

**src / componentes / TodosList / TodosList.js**

* // dependencias
* importación Reaccionar de 'reaccionar';

3

4 // dependencias locales

5 importación' ./TodosList.css';

6

7 por defecto de exportación ({niños}) => (

* <Ul>
* {niños}

10</ Ul>

11 );

Deconstruimos nuestra accesorios y seleccionar niños. No necesitamos ningún tipo de métodos de ciclo de vida, por lo que definimos como un componente puro de React.

También hemos importado TodosList.css, Que extrae los estilos pertinentes de

App.css:

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 55 |

**src / componentes / TodosList / TodosList.css**

* ul {
* list-style: none;

3margen: 50px auto;

4max-width: 800px;

5padding: 10px 15px;

6 }

Actualizar Aplicación usar TodosList:

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 TodosList importación de '../TodosList/TodosList';

7

8 ...

9

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <TodosList>
5. {\_.map (todos, (TODO, id) =>
6. <Li clave = {id}

|  |  |
| --- | --- |
| dieciséis | className = {todo.completed? 'Completado': ''} |
| 17 | onclick = {(E) => this.toggleTodo (id, e)}> |
| 18 | {} Todo.description |

1. </ Li>
2. )}
3. </ TodosList>
4. );
5. }

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 56 |

24

1. ...

Importamos nuestra TodosList y reemplazar nuestra ul con Lista de quehaceres la cual, por el momento, se comporta de manera muy similar a la anterior ul elemento html.

También eliminamos los estilos asociados con TodosList de App.css, Por lo que no tenemos ningún tipo de duplicación o estilo de enfrentamientos. Aquí está la actualizaciónApp.css sin el ul estilos:

**src / componentes / App / App.css**

* li {
* Antecedentes: #fafafa;

3border-radius: 5px;

4frontera: 1px solid # E1E1E1;

5box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

6color: # 888888;

7cursor: puntero;

8font-size: 2rem;

9margen: 10px 0;

1. padding: 15px 20px;
2. position: relative;
3. transición: todo 0.2s facilidad;
4. }

14

1. li: hover {
2. opacidad: 0,8;
3. }

18

1. .completed {
2. background: # E4E4E4;
3. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
4. Color: #AAAAAA;
5. text-decoration: tachado;
6. top: 3px;
7. }

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 57 |

**Construcción <TodosListItem />**

Vamos a crear un nuevo directorio para contener los archivos asociados con TodosListItem:

src /

componentes /

app /

TodosList /

TodosListItem /

TodosListItem.css

TodosListItem.js

TodosListItem.spec.js

index.js

Vamos a visitar TodosListItem.js:

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias
* importación Reaccionar de 'reaccionar';

3

4 // dependencias locales

5 importación' ./TodosListItem.css';

6

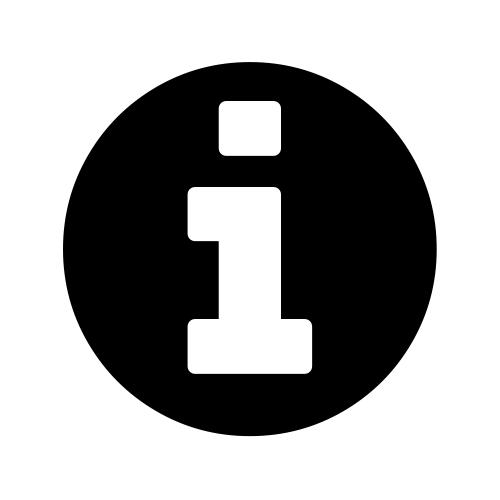
7 por defecto de exportación ({TODO, handleClick}) => (

8<ClassName li = {todo.completed? 'Completado': ''}

9onclick = {(E) => handleClick (e, todo.id)}>

1. {} Todo.description
2. </ Li>
3. );

Importamos el css asociado con nuestro componente, optar por la interfaz de componente puro de reaccionar y deconstruido los puntales.



Hemos eliminado la llave atribuir debido a que el bucle no sucede dentro del componente, sino más bien de consumo del componente.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 58 |

Por último, traemos lo largo de los estilos correspondientes a TodosListItem, Que son, por el momento, todos los estilos dentro de App.css.

**src / componentes / TodosListItem / TodosListItem.css**

* li {
* Antecedentes: #fafafa;

3border-radius: 5px;

4frontera: 1px solid # E1E1E1;

5box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

6color: # 888888;

7cursor: puntero;

8font-size: 2rem;

9margen: 10px 0;

1. padding: 15px 20px;
2. position: relative;
3. transición: todo 0.2s facilidad;
4. }

14

1. li: hover {
2. opacidad: 0,8;
3. }

18

1. .completed {
2. background: # E4E4E4;
3. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
4. Color: #AAAAAA;
5. text-decoration: tachado;
6. top: 3px;
7. }

La actualización de Let app.js para reflejar estos cambios:

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 59 |

**src / componentes / App / app.js**

* dependencias

...

* dependencias locales

...

TodosListItem importación de '../TodosListItem/TodosListItem';

...

render () {

const {todos} = this.state;

regreso (

<TodosList>

{\_.map (todos, (TODO, id) =>

<TodosListItem clave = {id}

TODO = {TODO}

handleClick = {this.toggleTodo.bind (este)} />

)}

</ TodosList>

);

}

...

Pasamos a la necesaria accesorios a TodosListItem y enlazar nuestra handleClickmétodo para el contexto correcto. Es la mejor práctica para unir estas devoluciones de llamada de una sola vez. Puede hacerlo ya sea en el constructor o en algún lugar apropiado. También podemos hacer ES6 + preajustes uso de Babel para poderse unen automático nuestros métodos.

Vamos a ver cómo se ve que:

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 60 |

**src / componentes / App / app.js**

...

~~toggleTodo (id, e) {~~

...

}

**toggleTodo = (id, e) => {**

**...**

**};**

...

Es una diferencia sutil, pero esta sintaxis proporcionada por Babel automático se une nuestros métodos de nuestra clase. Por último, hay que actualizar la referencia prop:

**src / componentes / App / app.js**

...

<TodosListItem clave = {} todo.id

TODO = {TODO}

handleClick = {} this.toggleTodo />

...

He encontrado esta sintaxis elegante y preferible a la :: sintaxis o de unión dentro del constructor.

Otro patrón He empezado a utilizar son [funciones de orden superior](http://eloquentjavascript.net/05_higher_order.html)31cuando se necesitan argumentos adicionales (tales como el índice de un elemento en un bucle). Por ejemplo, podemos actualizar nuestro método a lo siguiente:

1. <http://eloquentjavascript.net/05_higher_order.html>

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 61 |

**src / componentes / App / app.js**

...

toggleTodo = (id, e) => {

...

};

**toggleTodo = (id) => (e) => {**

**...**

**};**

...

La primera vez que nuestro nuevo toggleTodo se vuelve a llamar, volvemos una nueva función a la espera de ser llamado con mi como el argumento y el TODO carné de identidad almacenado en un cierre.

Necesitamos actualizar la forma en que llamamos nuestro nuevo método dentro de TodosListItem:

**src / componentes / TodosListItem / TodosListItem.js**

...

~~por defecto de exportación ({TODO, handleClick}) => (~~

<ClassName li = {todo.completed? 'Completado': ''}

onclick = {(E) => handleClick (e, todo.id)}> {} todo.description

</ Li>

~~);~~

**por defecto de exportación ({TODO, handleClick}) => (**

**<ClassName li = {todo.completed? 'Completado': ''} onclick = {handleClick (todo.id)}>**

**{} Todo.description**

**</ Li>**

**);**

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 62 |

...

Abra su navegador y podrá ver todo funcionando.

**Considerando U & I Conceptos**

Hasta ahora, hemos cubierto algunas de las mejores prácticas que cubren la encapsulación de componentes y la composición, por la co-localización de código relacionado con los componentes y tener “centrado” compo-nentes.

Eso está muy bien, pero nuestros estilos son globales y destinada al fracaso, ya que escalar nuestra aplicación. Aquí es donde reside el verdadero dolor. Vamos a tener que probar una nueva estrategia que nos guíe hacia la construcción de componentes U & I.

**Nombre Componentes espaciado**

Si aún no lo ha comenzado a pensar en cómo nombrar componentes espaciales, yo le animo a hacerlo. Hay muchas técnicas útiles que ayudan a mantener los estilos CSS choquen pero sólo unos pocos funcionan bien con estilos co-localizados.

**Nombre de la aplicación espaciado**

Una técnica poderosa es el nombre espaciamiento toda su aplicación. Es decir, todos los componentes dictada en su aplicación es un hijo de la aplicación de la raíz, lo que puede aumentar la especificidad de CSS por su nombre espaciado con un identificador único. Esto es muy bueno para ayudar a evitar situaciones en las que la aplicación utiliza una interfaz de usuario externo que pueda entrar en conflicto con sus selectores CSS.

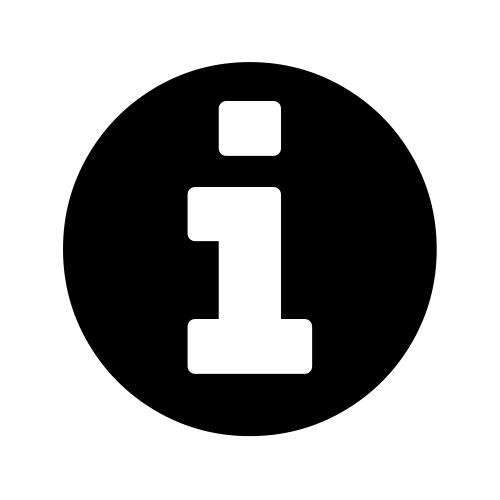
**Nombre del Componente espaciado**

Es posible que ya esté familiarizado con las convenciones de nombres, tales como [BEM](http://getbem.com/)32, [TRAJE](https://github.com/suitcss/suit/blob/master/doc/naming-conventions.md)33, Etc., que jugaría bien con [CEM (modificador Componente de elemento)](https://atendesigngroup.com/blog/component-element-modifier-design-pattern)34. No utilizaremos

1. <http://getbem.com/>
2. <https://github.com/suitcss/suit/blob/master/doc/naming-conventions.md>
3. <https://atendesigngroup.com/blog/component-element-modifier-design-pattern>

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 63 |

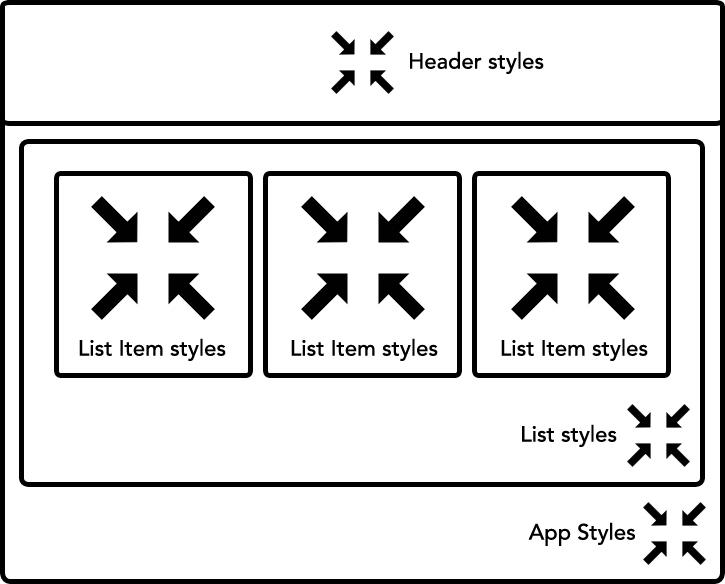
BEM aquí, pero puede decidir seguir BEM como una convención de nomenclatura para mitigar nombre de CSS colisión. Desde BEM está bien cubierta, vamos a utilizar una versión más simple de la CEM, dado que nuestra aplicación es muy magra.



Es más semántica a los componentes de espacio de nombres con clases CSS en lugar de identificadores. Puesto que los componentes, por naturaleza, pueden ser prestados varias veces dentro de una sola aplicación, clases CSS son semánticamente más apropiado.

**estilo unidireccional**

Si usted tiene alguna experiencia con Reaccionar o cualquiera de sus administradores estatales populares, deberá conocer el principio de [de datos unidireccional](http://redux.js.org/docs/basics/DataFlow.html)35. Ya hemos visto esto en acción al no permitir que los componentes secundarios, como la nuestraTodosListItem, Para mutar el estado de su matriz directamente, sino más bien por tener una interfaz para invocar una devolución de llamada para llevar a cabo esas operaciones. De esta manera, nuestraTodosListItemes simplemente una representación de los datos. Como las actualizaciones de datos, que fluya hacia abajo, a los componentes hijos, de un modo unidireccional de arriba hacia abajo. Del mismo modo, si tratamos a nuestros estilos como “datos”, podemos emplear una estrategia similar a la que nuestros estilos fluyen “hacia abajo”, como se ve aquí:



1. <http://redux.js.org/docs/basics/DataFlow.html>

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 64 |

Como se puede ver en el diagrama anterior, se debe apuntar para estilos de ser “intra-centrado” y en cascada hacia abajo. En principio, hay que evitar cualquier tipo de estilos que se extienden más allá de sus alcances.

**extensibilidad**

Siguiendo una estrategia de diseño de componentes rigurosa se presta bien para los consumidores de componentes sean capaces de ampliar o modificar el funcionamiento interno de un componente. Una de las peores situaciones en el desarrollo de la interfaz de usuario es el uso de elementos de la interfaz que son extremadamente difíciles de modificar o son inherentemente “bloqueados”. Cuando la construcción de un componente U & I, tenemos que diseñar con empatía, considerando los consumidores del componente.

**En acción**

Ahora, en la parte más divertida, donde se llega a aplicar estos conceptos.

**La implementación de la aplicación Nombre espaciado**

Ya ha dado cuenta de que estamos montaje de toda nuestra aplicación para un elemento HTML con un ID de raíz. Vamos a aprovechar lo que ya tenemos y simplemente hacer un cambio semántico.

**pública / index.html**

...

<Div id = "raíz"> </ div>

**<Div id = "todos-app"> </ div>**

...

Nombramos espacio de la aplicación con Todos aplicación Reaccionar donde toda nuestra aplicación está montado en el DOM.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | sesenta y cinco |

**src / index.js**

...

hacer(

<App />,

document.getElementById ( 'todos-aplicación')

);

Actualizar punto de entrada de la aplicación, asegurando que estamos enfocando el elemento HTML de montaje correcta. Todo lo demás se mantiene igual. Hemos introducido simplemente un cambio de nombre de semántica espaciamiento de toda nuestra solicitud bajo un id CSS identificable.

Usted puede preguntarse que dentro de nuestra styles.css, Nos estamos enfocando muchos elementos HTML globales. Eso está perfectamente bien! Una vez más, una vez que tenemos en cuenta los criterios de estilo unidireccionales, rápidamente vemos que estos estilos se aplican a todos los elementos dentro de nuestra aplicación. Este tipo de restablecimientos y estilos globales son comunes en las aplicaciones, por lo que es ideal para mantener sentados en la parte superior de nuestra aplicación.

**Ejecución Nombre del Componente espaciado**

Vamos a ver cómo otros conceptos U & I obtener aplicado a nuestros componentes.

Nuestra index.js y styles.csspermanece igual. Por lo tanto, vamos a pasar a nuestro primer componente:Aplicación:

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 clase predeterminada de exportación de aplicaciones se extiende Componente {

8

9Constructor (... args) {

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 66 |

1. ...
2. }

12

1. componentDidMount () {
2. ...
3. }

dieciséis

1. toggleTodo = (id) => (e) => {
2. ...
3. };

20

1. generateTodosListItem = (TODO, id) => (
2. <TodosListItem clave = {id}

|  |  |
| --- | --- |
| 23 | TODO = {TODO} |
| 24 | handleClick = {} this.toggleTodo /> |

1. );
2. render () {
3. const {todos} = this.state;
4. regreso (
5. <Div className = "aplicación">
6. <TodosList>
7. {\_.map (todos, this.generateTodosListItem)}
8. </ TodosList>
9. </ Div>
10. );
11. }
12. }

Nuestro principal Aplicación componente permanece casi igual, excepto por dos cosas:

* Una función para generar nuestra TodosListItem con el fin de mantener nuestra hacer Método magra.
* Anidación de nuestros componentes dentro de un nivel superior div con el nombre de la clase apropiada

aplicación.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 67 |

Creamos otra función para generar nuestra TodosListItem para mantener nuestra hacerMétodo magra. Puede llegar a ser difícil la lectura y la depuración grandehacer métodos.

Vamos a visitar nuestro segundo componente: TodosList.

**src / componentes / TodosList / TodosList.js**

* // dependencias
* Reaccionar importación, {} PropTypes de 'reaccionar';

3

4 // dependencias locales

5 importación' ./TodosList.css';

6

7 const TodosList = ({niños}) => (

8<Ul className = "todos-list">

* {niños}

10</ Ul>

11 );

12

13 TodosList.propTypes = {

14hijos: PropTypes.oneOfType ([

1. PropTypes.array,
2. PropTypes.element
3. ])
4. };

19

1. TodosList predeterminado de exportación;

Nuestra TodosList permanece casi igual, excepto que se añade una Todos listaclase de los espacios de nombres de nuestro componente. Además, siempre es una buena idea para realizar algunapropTypes validaciones.

la actualización TodosList estilos:

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 68 |

**src / componentes / TodosList / TodosList.css**

* # Todos-app-.todos lista {

2list-style: none;

3margen: 50px auto;

4max-width: 800px;

5padding: 10px 15px;

6 }

Sí, muy simple! En lugar de aplicar los estilos deul, Nos ha orientado nuestro componente por primera focalización nuestra aplicación carné de identidad seguido por el componente clase.

Para nuestro último componente, TodosListItem, Necesitaremos una nueva dependencia, [clase-nombres](https://github.com/JedWatson/classnames)36, Para ayudar a manejar estilos condicionales de una manera más elegante. Siga adelante y instalar esta nueva dependencia:

$ NPM instalan classnames --save

Ahora podemos visitar nuestra TodosListItem componente:

**src / componentes / TodosListItem.js**

* // dependencias
* \_ importar de 'lodash';
* classnames importación de classnames '';

4 PropTypes de importación de 'prop-tipos';

5 importación Reaccionar de 'reaccionar';

6

7 // dependencias locales

8 importación' ./TodosListItem.css';

9

1. const TodosListItem = ({TODO, handleClick}) => (
2. <Li className = {nombres de las clases ( 'todos-list-item', {
3. completado: todo.completed
4. <https://github.com/JedWatson/classnames>

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 69 |

1. })}
2. onclick = {handleClick (todo.id)}>
3. {} Todo.description
4. </ Li>
5. );

18

1. TodosListItem.defaultProps = {
2. que hacer: {},
3. toggleTodo: \_.noop
4. };

23

1. TodosListItem.propTypes = {
2. TODO: PropTypes.shape ({
3. completado: PropTypes.boolean,
4. Descripción: PropTypes.string,
5. ID: PropTypes.number
6. }),
7. handleClick: PropTypes.func
8. };

32

1. TodosListItem predeterminado de exportación;

Algunas cosas se han actualizado:

* Nombramos espacio de nuestro componente con una clase CSS.
* Importados nuestro nuevo classnamesmódulo. Este módulo permite útil para nosotros definir nombres de las clases complejas de una manera intuitiva. Por ejemplo, enhacer, Se aplica el Todos-list-item clase sin tener en cuenta, sin embargo, si el todo.completed estado es cierto, Añadimos también la terminadoclase. Es decir, si se ha completado el TODO, se poneTodos Elemento de lista de completado; en caso contrario, sólo seTodos-list-itemes regresado. Puede, por supuesto, añadir muchos más condicionales para los nombres de las clases más complejas. Puede leer más sobreclassnames en su [documentación](https://github.com/JedWatson/classnames)37.

1. <https://github.com/JedWatson/classnames>

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 70 |

* Establecimos defaultProps para asegurar manera tener valores sanos prop cuando TodosLis-TItem se rindió. los NOOP función es una función no-operación que vamos a uso, por lo que no se deje un error si no hay handleClick se proporciona función.
* Al igual que antes, validamos nuestros apoyos con propTypes validaciones.

Por supuesto, este componente no estaría completa hasta que actualizó su css archivo.

**src / componentes / TodosListItem / TodosListItem.css**

* # Todos-app .todos-list-item {

2Antecedentes: #fafafa;

3border-radius: 5px;

4frontera: 1px solid # E1E1E1;

5box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

6color: # 888888;

7cursor: puntero;

8font-size: 2rem;

9margen: 10px 0;

1. padding: 15px 20px;
2. position: relative;
3. transición: todo 0.2s facilidad;
4. }

14

1. # Todos-app .todos Elemento de lista de vuelo estacionario: {
2. opacidad: 0,8;
3. }

18

1. # Todos-app-.todos lista item.completed {
2. background: # E4E4E4;
3. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
4. Color: #AAAAAA;
5. text-decoration: tachado;
6. top: 3px;
7. }

Al igual que antes, nos dirigimos a nuestros elementos a través de la aplicación carné de identidad seguido por el componente clase.

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 71 |

**La implementación de Estilos extensibles y unidireccionales**

Puede parecer que hemos terminado, sin embargo, se nos olvidó dos conceptos importantes: estilos extender-able y unidireccionales. Puedes ver cual de estilos componente se extiende más allá de su alcance?

los margens Ajuste de TodosList y TodosListItemse extiende más allá de su propio alcance. Es decir, que afectan al comportamiento fuera de sí mismos. Puede parecer tonto a destacar este hecho en una pequeña aplicación, pero los componentes puede entrar en conflicto si no se sigue el contrato que ha de salir a cumplir en el primer lugar. Puede ser perfectamente razonable para mantenerlos tal cual, sin embargo, vamos a pasar por el ejercicio para ver cómo nos gustaría resolver este problema. Ya que estamos todavía en la tierra CSS, que en cascada sobre nuestros estilos de nuestra interfaz espaciadas nombre del componente.

Vamos a empezar desde los primeros hijos:

**src / componentes / TodosListItem / TodosListItem.css**

* # Todos-app .todos-list-item {

2Antecedentes: #fafafa;

3border-radius: 5px;

4frontera: 1px solid # E1E1E1;

5box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

6color: # 888888;

7cursor: puntero;

8font-size: 2rem;

9margen: 10px 0;

1. padding: 15px 20px;
2. position: relative;
3. transición: todo 0.2s facilidad;
4. }

14

1. # Todos-app .todos Elemento de lista de vuelo estacionario: {
2. opacidad: 0,8;
3. }

18

1. # Todos-app-.todos lista item.completed {
2. background: # E4E4E4;

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 72 |

1. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
2. Color: #AAAAAA;
3. text-decoration: tachado;
4. top: 3px;
5. }

Eliminamos la margen de TodosListItem.css e incluirlo en TodosList:

**src / componentes / TodosList / TodosList.css**

* # Todos-app-.todos lista {

2list-style: none;

3padding: 10px 15px;

4margen: 50px auto;

5max-width: 800px;

6 }

7

8 **# Todos-app-.todos lista .todos-list-item {**

9**margen: 10px 0;**

1. **}**

También eliminamos la anchura máxima y margen de TodosList e incluirlo en el componente de los padres:

**src / componentes / App / App.css**

* .app # todos aplicación {

2

3 }

4

5 # todos aplicación .app .todos-list {

6margen: 50px auto;

7max-width: 800px;

8 }

Si visita el navegador, toda la aplicación puede parecer lo mismo, pero la verdad es que ha sido fuertemente refactorizado a una T & I centrado fundación.

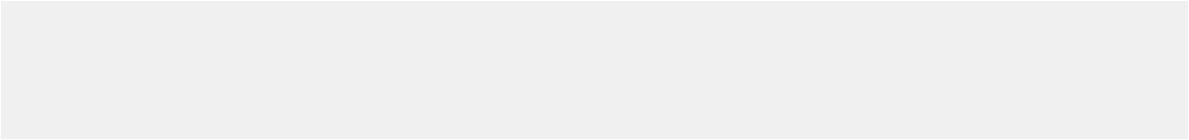
|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 73 |

**Resumen**

En este capítulo, hemos descrito algunos conceptos que son un buen punto de partida en el viaje de U & I. A pesar de que estos son grandes prácticas para la construcción de componentes U & I, no son, técnicamente, a prueba de balas. Es decir, podemos todavía tenga problemas con colisiones de estilo.

En general, proporcionan una base sólida para abordar algunos de los criterios de U & I. Vamos a echar un vistazo rápido a dónde estamos en relación con el T & I contratamos nos define enCapítulo 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **CSS** |  |
|  |  |  |  |
|  | **No hay espacio de nombres global** | \* |  |
|  | **estilos unidireccionales** | ✔ |  |
|  | **eliminación de código muerto** |  |  |
|  | **minificación** |  |  |
|  | **Las constantes se pueden compartir** |  |  |
|  | **resolución determinista** | \* |  |
|  | **Aislamiento** | \* |  |
|  | **extensible** | \* |  |
|  | **documentable** | N / A |  |
|  | **Presentable** | N / A |  |
|  | |  |  |
| [ ✔ Cumplido] [\* Pseudo cumplido] | |  |  |
|  |  |  |  |



Algunos de ellos son seudocompletado, lo que significa que no cumplen completamente los criterios, pero son lo suficientemente cerca de directrices claras dadas y las mejores prácticas. Además, no vamos a cubrir documentable y presentable hasta los últimos capítulos, lo que vamos a hacer caso omiso de ellos por el momento.

A pesar de que no fueron capaces de cumplir con todos nuestros criterios de U & I y algunos eran solamente

|  |  |
| --- | --- |
| Capítulo 5: Uso de U & I Conceptos | 74 |

*seudo* cumplido, esto puede ser más que suficiente para muchos requisitos U & I. Desde mi experiencia, siguiendo estos principios básicos ya está muy por delante de las muchas aplicaciones que he encontrado que contienen alto grado de dependencia entre los componentes con los estilos que se extienden más allá de su alcance.

¡Uf! Hemos terminado! Bueno, algo así ... He aquí un resumen rápido de lo que hemos cubierto en este capítulo:

* Modularizar sus componentes.
* Construir componentes componibles cuando sea apropiado.
* Nombre del espacio, de forma inteligente. Es decir, aumentar su especificidad CSS y traer un poco de estructura a su aplicación por el nombre espaciar su aplicación y componentes.
* Mantener su hacer métodos lean y se centró, por lo que es fácil de leer y depurar.
* Mantenga sus componentes enfocados, funcional y sostenible.
* Asegúrese de definir y proteger su API de los componentes con los apoyos por defecto y validación de tipo hélice.
* Lo más importante es pensar en sus estilos de componentes de forma unidireccional. Es decir, estilos deben fluir hacia abajo, de padres a hijos. Debemos tratar de evitar estilos de componentes que se extienden más allá de su alcance, a menos que no hay absolutamente buena razón para ello.

Vamos a pasar a nuestro siguiente capítulo para ver cómo podemos limpiar aún más nuestros componentes.

**Capítulo 6: Explorando Preprocesadores CSS**

En el capítulo anterior hemos cubierto algunas de las mejores prácticas U & I que nos guíen en la refactorización nuestra aplicación. En este capítulo, vamos a aprovechar una preprocesador CSS para ayudar a adherirnos a esos principios.

**¿Qué es un preprocesador de CSS?**

Un preprocesador es un programa transforma un tipo de datos a otro. En el caso de CSS, podemos aprovechar un pre-procesadores para extender CSS con variables, operadores, funciones, interpolaciones mixins y muchos otros activos más utilizables.

**¿Por qué utilizar un preprocesador?**

CSS es primitiva e incompleta. preprocesadores CSS nos ofrecen muchas características útiles y avanzadas que ayudan a conseguir escritura de código reutilizable, mantenible y extensible en el CSS. Mediante el uso de un preprocesador, puede fácilmente aumentar su productividad y reducir la cantidad de código que está escribiendo.

**Conoce Sass**

Aunque hay muchos preprocesadores CSS de gran alcance, tales como [Aguja](http://stylus-lang.com/)38, [Menos](http://lesscss.org/)39, [Mito](http://www.myth.io/)40, Etc., que va a utilizar [Hablar con descaro a](http://sass-lang.com/)41. Sass ha crecido en popularidad para una variedad de

1. <http://stylus-lang.com/>
2. <http://lesscss.org/>
3. <http://www.myth.io/>
4. <http://sass-lang.com/>

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 76 |

razones - más notablemente en Bootstrap bajado menos en favor de su liberación con Sass Bootstrap 4. Dicho esto, no dude en utilizar cualquier preprocesador CSS que se siente más cómodo, dado que ofrecen características similares con muy poca diferencia en la sintaxis y el estilo.

**Sass en Acción**

Como nos Bootstrapped nuestra aplicación usando Create-reaccionar aplicación, Tendremos que modificar nuestra webpack a ser capaz de soportar .scssarchivos. Esta es una de las razones por las que ejecuta elexpulsar guión, de manera que eventualmente podríamos modificar Create-reaccionar aplicación'S config.

Instalar un webpack hablar con descaro a cargador:

$ NPM instalar Sass-loader nodo-Sass --save-dev

Sólo nos instalamos estas dependencias como dev-dependencias, Ya que sólo son necesarios en nuestra línea de acumulación. Una vez instalado, se pueden encontrar los archivos de configuración en el interior webpack/ config. Sólo estamos preocupados condesarrollo, Pero se puede seguir una serie de pasos similar para otros entornos.

**config / webpack.config.dev.js**

...

{

Prueba: /\.css$/,

**Prueba: /\.(css|scss)$/,**

utilizar: [

require.resolve ( 'estilo-loader'),

{

loader: require.resolve ( 'css-loader'),

Opciones: {

importLoaders: 1,

},

},

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 77 |

{

loader: require.resolve ( 'postcss-loader'),

Opciones: {

ident: 'postcss',

plugins: () => [

require ( 'postcss-flexbugs-correcciones'),

autoprefixer ({

navegadores: [

'> 1%',

'últimos 4 versiones',

'Firefox ESR',

'No es decir, <9',

],

FlexBox: 'no-2009',

}),

],

},

},

**require.resolve ( 'Sass-loader')**

],

},

...

Su configuración webpack puede variar ligeramente, pero en esencia, que tendrá que encontrar y actualizar su gestor de CSS para ser capaz de ejecutar más de los cargadores apropiados tanto css y .scss archivos.

Vaya por delante y reinicie su proyecto mediante la ejecución inicio NPMy todo debería funcionar igual que antes. Ahora, es hora de refactorizar nuestros estilos, componente a componente, mediante el aprovechamiento de Sass.

Si esto no funciona con tu versión de Create-reaccionar aplicación, Puede intentar lo siguiente:

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 78 |

**config / webpack.config.dev.js**

{

excluir: [

/\.html$/,

/\.(js|jsx)(\?.\*)?$/,

/\.css$/,

**/\.(css|scss)$/,**

/\.json$/,

/\.svg$/

],

cargador: 'url',

consulta: {

límite: 10000,

Nombre: 'estática / media / [nombre] [picadillo: 8]. [ext]'

}

},

Si sigues teniendo problemas, por favor refiérase a la del libro [código fuente](https://github.com/FarhadG/ui-react)42 para inspeccionar las versiones de las dependencias y la configuración webpack utilizado.



Nosotros no vamos a utilizar todas las características que ofrece Sass, pero vamos a cubrir algunas de sus funciones fundamentales. Por ejemplo, una técnica poderosa para aumentar la especificidad de CSS anida. Por supuesto, que no necesariamente tiene que usar un preprocesador para hacer esto como se puede escribir las clases con CSS ol llano', pero eso es tedioso y propenso a errores.

**Configuración de los estilos**

Crear un estilos directorio para alojar archivos globales (por ejemplo, theme.scss, typography.scss, Etc.) en una aplicación de producción de grado.

1. <https://github.com/FarhadG/ui-react>

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 79 |

**src / estilos / theme.scss**

* $ APP: 'todos aplicación';

2

3 $ Principal-background-color: # F1F1F1;

4 $ Principal-font-family: "Helvetica Neue", "Arial", "sans-serif"; 5

6 $ Gris claro: #fafafa;

7 $-Gris oscuro: # 888888;

Utilizamos una variable para definir el nombre de nuestra aplicación, junto con algunas variables de color.

Nos movemos nuestra styles.css presentar en nuestra estilos directorio y renombrarlo como Glob-als.scss:

**src / estilos / globals.scss**

* html {
* caja de tamaño: Frontera de la caja;

3 }

4

5 \*, \*:antes después {

6cuadro de dimensionamiento: heredar;

7 }

8

9 cuerpo {

1. background: # F1F1F1;
2. font-size: 10px;
3. font-family: "Helvetica Neue", "Arial", "sans-serif";
4. margin: 0;
5. padding: 0;
6. }

Puesto que estos serán los estilos entrada para toda nuestra aplicación, vamos a crear un nuevo archivo, main.scss lo que importa todos en general .scss archivos para arrancar nuestra aplicación.

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 80 |

**src / estilos / main.scss**

* @import "theme.scss";

2 @import "globals.scss";

Desde que nos mudamos y renombrado styles.css, Importaremos main.scss para arrancar nuestra aplicación:

**src / index.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 ~~importación' ./styles.css';~~

7

8 **importación' ./styles/main.scss';**

9

1. ...

Dicho esto, realmente no hemos utilizado nuestra theme.scss archivos, por lo que vamos a utilizar dentro de globals.scss.

**src / estilos / globals.scss**

* ...

2

3 cuerpo {

4Antecedentes: $ principal-background-color;

5font-size: 10px;

6font-family: $ principal fuente de la familia;

7margin: 0;

8padding: 0;

9 }

Desde importamos main.scss archivo que contiene theme.scss dentro de su contexto, podemos hacer referencia a esas variables.

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 81 |

**Refactorización <app />**

Actualizar App.css a App.scss y actualizar nuestro importar declaración:

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 ~~importación' ./App.css';~~

7

8 **importación' ./App.scss';**

9

1. ...

A continuación, realizar los siguientes cambios en el estilo:

**src / componentes / App / App.scss**

* @import '../../styles/theme';

2

3 ## {} $ APP .app {

4

5.todos-lista {

6margen: 50px auto;

7max-width: 800px;

* }

9

1. }

Aquí es donde nuestro ID de aplicación se vuelve útil. importamostheme.scss y interpolamos nuestra $ APPvariable para espacio de nombres de nuestro componente. La sintaxis puede parecer un poco extraño, pero todo lo que estamos haciendo es establecer un ID (es decir,#) E interpolando nuestra $ APP variable con el #{VARIABLE}sintaxis. Ahora, podemos piel fácilmente toda nuestra aplicación en diferentes contextos cambiando temas con una variable.

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 82 |

**Refactorización <TodosList />**

Actualizar TodosList.css a TodosList.scss y actualizar nuestra declaración de importación:

**src / componentes / TodosList / TodosList.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 ~~importación' ./TodosList.css';~~

7

8 **importación' ./TodosList.scss';**

9

1. ...

A continuación, realizar los siguientes cambios en el estilo:

**src / componentes / TodosList / TodosList.scss**

* @import '../../styles/theme';

2

3 ## {} $ APP-.todos lista {

4list-style: none;

5padding: 10px 15px;

6

7&-ít {

8margen: 10px 0;

9}

1. }

Usamos el yoperador mantener nuestro código seco. losy operador en el interior de un selector concatena el nombre del padre en su lugar, por lo que nos queda .todos-list-item.

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 83 |

**Refactorización <TodosListItem />**

Actualizar TodosListItem.css a TodosListItem.scss y actualizar nuestro estado de importación-ment:

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 ~~importación' ./TodosListItem.css';~~

7

8 **importación' ./TodosListItem.scss';**

9

1. ...

A continuación, realizar los siguientes cambios en el estilo:

**src / componentes / TodosListItem / TodosListItem.scss**

* @import '../../styles/theme';

2

3 ## {} $ APP .todos-list-item {

4Antecedentes: $ gris claro;

5border-radius: 5px;

6frontera: 1px solid # E1E1E1;

7box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

8Color: gris oscuro $;

9cursor: puntero;

1. font-size: 2rem;
2. padding: 15px 20px;
3. position: relative;
4. transición: todo 0.2s facilidad;
5. Y: hover {

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 84 |

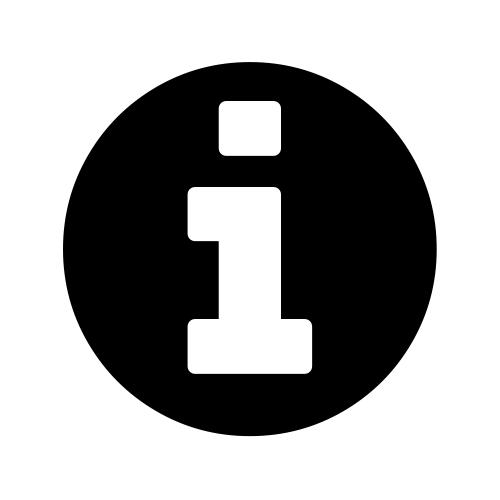
1. opacidad: 0,8;
2. }

18

1. Y .completed {
2. Antecedentes: se oscurecen ($ gris claro, 10);
3. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
4. Color: #AAAAAA;
5. text-decoration: tachado;
6. top: 3px;
7. }
8. }

Todos nuestros .todos-list-item estilo se anidan y hemos hecho uso de la y operador sea capaz de anidar adicional selectores vinculados a .todos-list-item.

También usamos el oscurecerfunción. En lugar de tratar con otro color CSS, podemos aprovechar operaciones útiles, tales como Sassoscurecer, Para modificar los colores existentes.



preprocesadores CSS, como Sass, tienen muchos operadores útiles que pueden hacer que el trabajo con los colores de una brisa. Asegúrese de aprovechar en sus proyectos.

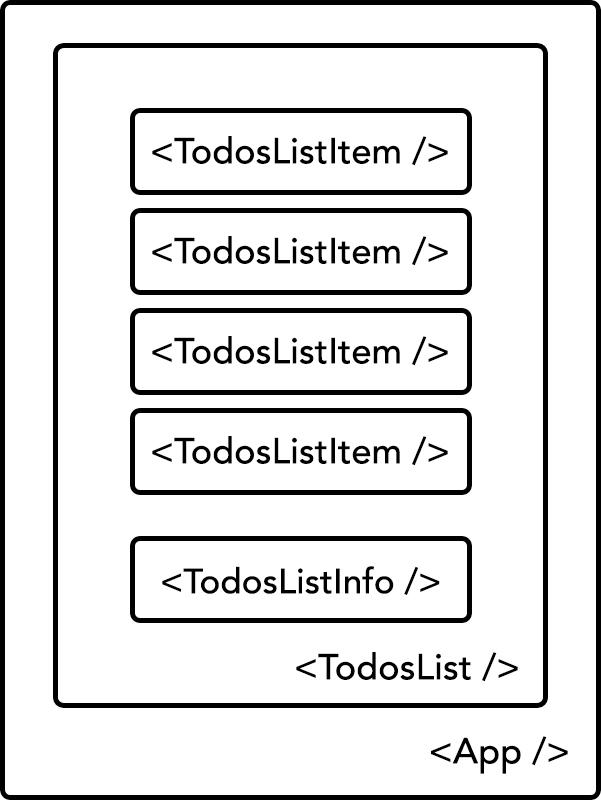
**mejoras**

Ahora que tenemos un buen flujo para la creación de componentes, vamos a introducir un componente adicional, como un medio para mostrar la facilidad de nuestra arquitectura existente.

**En <TodosListInfo />**

Se introduce un nuevo componente que se encuentra dentro de nuestra TodosList, Que proporciona información básica sobre el estado de nuestra todos.

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 85 |



**Diagrama de componentes**

**Construcción <TodosListInfo />**

Comience por crear nuestro directorio y los archivos de componente:

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 86 |

src /

componentes /

TodosListInfo /

TodosListInfo.scss

TodosListInfo.js

TodosListInfo.spec.js

A continuación, haremos un componente haciendo que el número de todos restantes sobre el número total de todos.

* // dependencias
* \_ importar de 'lodash';
* Reaccionar importación, {} PropTypes de 'reaccionar';

4

5 // dependencias locales

6 TodosListItem importación de '../TodosListItem/TodosListItem';

7 importación' ./TodosListInfo.scss';

8

9 const TodosListInfo = ({todos}) => {

1. const todosCount = \_.size (todos);
2. const completedTodosCount = \_ (todos) .Filter tamaño ( 'completado') (.);
3. regreso (
4. <Li className = "todos-list-info">
5. {CompletedTodosCount} / {} todosCount completado
6. </ Li>
7. );
8. };

18

1. TodosListInfo.defaultProps = {
2. Todos: []
3. };

22

1. TodosListInfo.propTypes = {
2. Todos: PropTypes.objectOf (TodosListItem.propTypes.todo)
3. };

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 87 |

26

1. por defecto de exportación TodosListInfo;

Por supuesto que necesitamos algunos estilos:

* @import '../../styles/theme';

2

3 ## $ {app} .todos-list-info {

4Color: gris oscuro $;

5font-size: 14px;

6text-align: right;

7 }

Utilizamos unas pocas funciones Lodash útil para filtrar y contar el número de todos. También importamosTodosListItemvolver a utilizar los mismos tipos prop definiciones de validación. Esto puede sugerir una profunda dependencia entre estos dos componentes diferentes, sino una única fuente de información para estas validaciones es una buena práctica.

Si estas dependencias se convierten en imposible de mantener, hay varios patrones que pueden ayudar a desacoplar los componentes entre sí:

1. Crear un archivo de entrada dentro de su componentes directorio que exporta todos los compo-nentes de una sola fuente.
2. Extraer todo componente propTypes en un recurso compartido como una única fuente de verdad.

Vamos a mantener las cosas simples en nuestro proyecto, pero es bueno estar al tanto de la gestión de la dependencia de su aplicación como el número de componentes U & I Grow.

Vamos a usar nuestro nuevo componente:

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 88 |

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 importación de TodosListInfo '../TodosListInfo/TodosListInfo';

7

8 clase App extiende Componente {

9

1. Constructor (... args) {
2. ...
3. }

13

1. componentDidMount () {
2. ...
3. }

17

1. toggleTodo = (id) => (e) => {
2. ...
3. };

21

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

25

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <Div className = "aplicación">
5. <TodosList>
6. {\_.map (todos, this.generateTodosListItem)}
7. <TodosListInfo todos = {todos} />
8. </ TodosList>
9. </ Div>
10. );

|  |  |
| --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS | 89 |

1. }
2. }
3. Aplicación por defecto de exportación;

Importamos nuestra TodosListInfo componente y lo hacen en el interior de TodosListcon los accesorios apropiados. Por ahora, las ventajas de desacoplamiento de componentes y garantizar una clara separación de funciones deben ser claros.

Abra su navegador y podrá ver nuestro nuevo componente de la prestación de tareas pendientes actualizaciones en tiempo real.

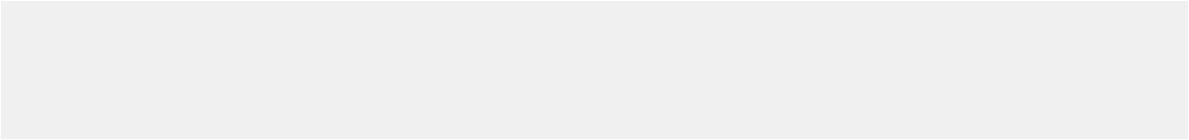


**Resumen**

En este capítulo, hemos introducido Sass, un preprocesador CSS, y aprovechamos algunas de las características comunes para ayudar a mantener algunas de las prácticas que se establece en los capítulos anteriores. También hemos añadido un nuevo componente como un medio para probar la simplicidad de la adición de un nuevo componente a nuestra suite U & I.

¿Dónde nos encontramos con nuestra lista de comprobación T & I?

|  |  |  |
| --- | --- | --- |
| Capítulo 6: Explorando Preprocesadores CSS |  | 90 |
|  | **CSS** | **SCSS** |
|  |  |  |
| **No hay espacio de nombres global** | \* | \* |
| **estilos unidireccionales** | ✔ | ✔ |
| **eliminación de código muerto** |  |  |
| **minificación** |  |  |
| **Las constantes se pueden compartir** |  |  |
| **resolución determinista** | \* | \* |
| **Aislamiento** | \* | \* |
| **extensible** | \* | \* |
| **documentable** | N / A | N / A |
| **Presentable** | N / A | N / A |
|  |  |  |
| [ ✔ Cumplido] [\* Pseudo cumplido] |  |  |
|  |  |  |



Nosotros no cumplir ciertos criterios adicionales, pero si el estado anterior contrato era satisfactoria, la adición de un preprocesador de CSS para el flujo de trabajo es una gran opción para la calidad general.

Vamos a pasar a nuestro siguiente capítulo, donde vamos más allá de las convenciones de nomenclatura y técnicas inteligentes para evitar el choque de estilos.

**Capítulo 7: Exploración de módulos CSS**

En el capítulo anterior hemos introducido Sass, un poderoso preprocesador CSS, en nuestra línea de desarrollo. Nos aprovechamos algunos Sass útiles características para ayudar a mantener algunos de nuestros requerimientos núcleo U & I. A pesar de que hemos mejorado la calidad de nuestros componentes U & I, los estilos CSS todavía pueden chocar en aplicaciones más complejas.

Si este es su preocupación, estás de suerte! En este capítulo, vamos a explorar[módulos CSS](https://github.com/css-modules/css-modules)43, Una nueva e interesante tecnología, que ofrece una mejor manera de mitigar el choque CSS.

**¿Cuáles son los módulos de CSS?**

De acuerdo con la [repositorio de módulos CSS](https://github.com/css-modules/css-modules)44:



archivos CSS en el que todos los nombres de las clases y los nombres están en el ámbito de la animación a nivel local por defecto. Módulos CSS es un paso en el proceso de construcción que cambia los nombres de clases y selectores para estar en el ámbito local.

Por ejemplo:

1. <https://github.com/css-modules/css-modules>
2. <https://github.com/css-modules/css-modules>

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 92 |

estilos de importación "./styles.css";

elemento const = () => (

<H1 className = {styles.title}>

¡Hola Mundo!

</ H1>

);

Durante nuestro proceso de construcción, el cargador de módulos CSS busca a través styles.css y hace que la .título clase accesibles a través styles.title. Detrás de las escenas, sin embargo, nuestra plantilla y estilos se generan con nuevos personajes que sustituye tanto a la clase HTML y la clase selector CSS.

Un ejemplo de lo que puede verse como:

<Clase h1 = "\_ styles\_\_title\_3095">

¡Hola Mundo!

</ H1>

.\_styles\_\_title\_3095 {

background-color: rojo;

}

El atributo de clase y el selector .títulose sustituyen por esta completamente nueva cadena única. En resumen, las clases se generan de forma dinámica, única y asignan a los estilos correctos.

**¿Por qué usar CSS módulos?**

Es una garantía de que todos los estilos para un solo componente viven en un lugar y están en el ámbito local. Este enfoque está diseñado para solucionar el problema del alcance global en CSS. módulos CSS permiten asignar un nombre a sus selectores CSS en cualquier forma o modalidad, sin necesidad de preocuparse por las colisiones de nombres. Con los módulos de CSS, y el concepto de ámbito local de forma predeterminada, se evita este problema.

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 93 |

**Módulos CSS en Acción**

Afortunadamente, con nuestra configuración actual webpack podemos permitir que los módulos CSS con facilidad. Vaya por delante y actualizar el archivo de configuración para permitir webpack módulos CSS:

**config / webpack.config.dev.js**

...

{

Prueba: /\.(css|scss)$/,

utilizar: [

require.resolve ( 'estilo-loader'),

{

loader: require.resolve ( 'css-loader'),

Opciones: {

importLoaders: 1,

**módulos: true,**

**localIdentName: '[nombre] \_ [locales] \_ [picadillo: base 64: 5]**

},

},

{

loader: require.resolve ( 'postcss-loader'),

Opciones: {

* Necesaria para la importación CSS externos para trabajar
* https: //github.com/facebookincubator/create-react-a \

pp / temas / 2677

ident: 'postcss',

plugins: () => [

require ( 'postcss-flexbugs-correcciones'),

autoprefixer ({

navegadores: [

'> 1%',

'últimos 4 versiones',

'Firefox ESR',

'No es decir, <9', // Reaccionar no soporta IE8 de todos modos

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 94 |

],

FlexBox: 'no-2009',

}),

],

},

},

require.resolve ( 'Sass-loader')

],

},

...

Nuestra css-loaderya cuenta con CSS funcionalidad de los módulos - que sólo tenía que poder en-él. Usted puede preguntarse lo que ellocalIdentName, [nombre], [local] y [de hash: base64: 5] representar. Simplemente estamos definiendo el nombre del selector CSS. Hay otras opciones que se pueden encontrar en el css-loaderdocumentación. Esta combinación es intuitiva paradesarrolloModo ya que es fácil de razonar acerca y depurar. Estamos generando nuestros selectores CSS como una combinación de su ruta, nombre de componente y una única base 64 5 dígitos cadena codificada atadas con\_. EnproducciónSin embargo, se puede omitir muchas de estas variables para derivar el selector CSS más corto.

Después de actualizar la configuración webpack, ninguno de los estilos funcionará. Eso es porquecss-loader está esperando por nosotros de trabajar con módulos de CSS.

**refactoring estilos / \*. SCSS**

Dado que todos nuestros estilos generales son globales por naturaleza, es decir, no hay nada específico para aplicación como una clase CSS, no es necesario hacer nada.

**Refactorización <app />**

Vamos a empezar con nuestra importar declaración:

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 95 |

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 estilos de importación de' ./App.scss';

7

8 clase App extiende Componente {

9

1. Constructor (... args) {
2. ...
3. }

13

1. componentDidMount () {
2. ...
3. }

17

1. toggleTodo = (id) => (e) => {
2. ...
3. };

21

1. generateTodosListItem = (TODO, id) => (
2. <TodosListItem clave = {id}

|  |  |
| --- | --- |
| 24 | TODO = {TODO} |
| 25 | pStyles = {} estilos |
| 26 | handleClick = {} this.toggleTodo /> |

1. );
2. render () {
3. const {todos} = this.state;
4. regreso (
5. <Div className = {} styles.app>
6. <TodosList pStyles = {} estilos>
7. {\_.map (todos, this.generateTodosListItem)}

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 96 |

1. <TodosListInfo todos = {todos} />
2. </ TodosList>
3. </ Div>
4. );
5. }
6. }

41

1. Aplicación por defecto de exportación;

Se introduce un nuevo apoyo, pStyles abreviatura de propStyles, para ser capaz de pasar nuestros estilos abajo.

**src / componentes / App / App.scss**

* @import '../../styles/theme';

2

3 .app {}

4

5 .todosList {

6margen: 50px auto;

7max-width: 800px;

8 }

9

1. .todosListItem {
2. margen: 10px 0;
3. }

Dado que todos nuestros estilos están en el ámbito local, ya no podemos tener nuestro TodosListSu estilo en cascada hacia abajo a sus hijos. Nos burbujea los estilos de un nivel superior aAplicación, De manera que podamos pasarlos abajo según sea necesario. Esta no es la solución más elegante, ya que:

* Podemos importar los estilos asociados con todosListItem dentro de nuestra A-dosListItem.scss y tienen módulos CSS crean el ámbito local para ellos.
* Podemos utilizar el compone palabra clave y recuperar esos estilos.

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 97 |

No vamos a perseguir estas opciones, ya que crean una profunda dependencia entre compo-nentes. Esto puede ser perfectamente apropiado en sus aplicaciones, sin embargo, vamos a optar por laapuntalar Interfaz para el paso de los estilos abajo, ya que vamos a utilizar esto como una base para futuros capítulos.

El ID de aplicación se ha eliminado y clases CSS se han actualizado para estar en formato camelCase, por lo que podemos hacer referencia a ellos apropiadamente dentro de nuestras plantillas. Podríamos seguir manteniendo nuestros nombres de las clases dekebab de los casos formatear y hacer referencia a ellos a través de estilos [ 'todos-list'], Pero yo prefiero el Javascript el caso de Carmel convención, dado que estamos escribiendo la mayor parte de nuestra aplicación en JS.

**Refactorización <TodosList />**

Vamos a empezar por la actualización de cómo importar y aplicar nuestros estilos:

**src / componentes / TodosList / TodosList.js**

* // dependencias
* classnames importación de classnames '';
* Reaccionar importación, {} PropTypes de 'reaccionar';

4

5 // dependencias locales

6 estilos de importación de' ./TodosList.scss';

7

8 const TodosList = {(niños, pStyles}) => (

9<ul className = {nombres de las clases (

1. styles.todosList,
2. pStyles.todosList
3. )}>
4. {niños}
5. </ Ul>
6. );

dieciséis

1. TodosList.defaultProps = {
2. niños: [],
3. pStyles: {
4. todosList: ''

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 98 |

1. }
2. };
3. TodosList.propTypes = {
4. hijos: PropTypes.oneOfType ([
5. PropTypes.array,
6. PropTypes.element
7. ]),
8. pStyles: PropTypes.shape ({
9. todosList: PropTypes.string
10. })
11. };

33

1. TodosList predeterminado de exportación;

Importamos nuestros estilos con un nombre, estilos, Por lo que podemos hacer referencia a sus llaves. Usamosclassnames añadir base y apuntalar estilos para nuestro elemento. Por último, hemos actualizado nuestra prop definiciones para reflejar estos cambios.

Ahora nuestros estilos actualizados:

**src / componentes / TodosList / TodosList.scss**

* @import '../../styles/theme';

2

3 .todosList {

4list-style: none;

5padding: 10px 15px;

6 }

**Refactorización <TodosListInfo />**

¡Enjuague y repita! Vamos a actualizar nuestra plantilla:

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 99 |

**src / componentes / TodosListInfo / TodosListInfo.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 estilos de importación de' ./TodosListInfo.scss';

7

8 const TodosListInfo = ({todos}) => {

9const todosCount = \_.size (todos);

1. const completedTodosCount = \_ (todos) .Filter tamaño ( 'completado') (.);
2. regreso (
3. <Li className = {styles.todosListInfo}>
4. {CompletedTodosCount} / {} todosCount completado
5. </ Li>
6. );
7. };

17

1. ...

Hacemos referencia a nuestros estilos y aplicar el todosListInfo clase para nuestro componente.

**src / componentes / TodosListInfo / TodosListInfo.scss**

* @import '../../styles/theme';

2

3 .todosListInfo {

4Color: gris oscuro $;

5font-size: 14px;

6text-align: right;

7 }

Podríamos proporcionar una pStylesinterfaz, pero vamos a seguir este componente encerrado en su formato actual. ¿Por qué? Porque con módulos CSS, podemos ...

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 100 |

**Refactorización <TodosListItem />**

¡Enjuague y repita! Vamos a actualizar nuestra plantilla:

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias
* \_ importar de 'lodash';
* classnames importación de classnames '';

4 PropTypes de importación de 'prop-tipos';

5 importación Reaccionar de 'reaccionar';

6

7 // dependencias locales

8 estilos de importación de' ./TodosListItem.scss';

9

1. const TodosListItem = ({pStyles, todo, handleClick}) => (
2. <li
3. className = {nombres de las clases (
4. styles.todosListItem,
5. pStyles.todosListItem,
6. {
7. [Styles.completedTodosListItem]: todo.completed,
8. [PStyles.completedTodosListItem]: todo.completed
9. }
10. )}
11. onclick = {handleClick (todo.id)}>
12. {} Todo.description
13. </ Li>
14. );

24

1. TodosListItem.defaultProps = {
2. que hacer: {},
3. toggleTodo: \_.noop,
4. pStyles: {
5. todosListItem: '',
6. completedTodosListItem: ''

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 101 |

1. }
2. };
3. TodosListItem.propTypes = {
4. handleClick: PropTypes.func,
5. TODO: PropTypes.shape ({
6. completado: PropTypes.boolean,
7. Descripción: PropTypes.string,
8. ID: PropTypes.number
9. }),
10. pStyles: {
11. todosListItem: PropTypes.string,
12. completedTodosListItem: PropTypes.string
13. }
14. };
15. TodosListItem predeterminado de exportación;

Nombramos nuestra estilos y que se refiere a nuestra todosListItemcomo un estilo de base. Si el artículo esterminado, Se aplica el completedTodosListItemestilos. Actualizamos nuestras definiciones de tipo calzo para las diversas definiciones de estilo este componente apoya.



el ES6 { [valor variable } sintaxis permite interpolar variables en clave de un objeto.

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 102 |

**src / componentes / TodosListItem / TodosListItem.scss**

* @import '../../styles/theme';

2

3 .todosListItem {

4Antecedentes: $ gris claro;

5border-radius: 5px;

6frontera: 1px solid # E1E1E1;

7box-shadow: 0 2px 1px 0 RGBA (0, 0, 0, 0,2);

8Color: gris oscuro $;

9cursor: puntero;

1. font-size: 2rem;
2. padding: 15px 20px;
3. position: relative;
4. transición: todo 0.2s facilidad;
5. Y: hover {
6. opacidad: 0,8;
7. }
8. }
9. .completedTodosListItem {
10. Antecedentes: se oscurecen ($ gris claro, 10);
11. box-shadow: inserción 0 1px 2px 0 RGBA (0, 0, 0, 0,3);
12. Color: #AAAAAA;
13. text-decoration: tachado;
14. top: 3px;
15. }

Nos aseguramos de nuestro partido nombres de las clases y estamos listo para salir.

Si visita el navegador, todo debería funcionar como se espera. Todos los estilos necesarios se hayan aplicado y nuestra aplicación funciona perfectamente, gracias a los módulos de CSS. Esto puede parecer trivial, pero es bastante sorprendente! Ya no tenemos selectores globales.

|  |  |
| --- | --- |
| Capítulo 7: Exploración de módulos CSS | 103 |



**Ejercicio sugerido**

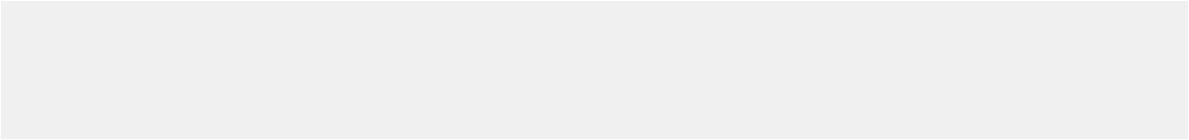
módulos CSS son extremadamente potente y os animo a explorar estas suge-ciones antes de continuar:

* inspeccionar el DOMy mirar los selectores CSS aplicados. Experimento con la configuración webpack y explorar los resultados.
* Tira un depurador o consola Estado dentro de su componente y observe el estilosarchivo. Esto le ayudará a comprender cómo se generan las clases CSS.
* Prueba las diferentes estrategias para pasar alrededor de estilos.
* Experimento con diferentes maneras de aplicar estilos a componentes: estructura plana frente a la estructura anidada, funcional específico frente a nombres específicos de componentes (por ejemplo, todosListItem versus que hacer).

**Resumen**

En este capítulo, convertimos toda nuestra aplicación para utilizar los módulos de CSS. Vamos a ver cómo esta estrategia tarifas contra los otros que hemos tratado en los capítulos anteriores:

|  |  |  |  |
| --- | --- | --- | --- |
| Capítulo 7: Exploración de módulos CSS |  |  | 104 |
|  | **CSS** | **SCSS** | **Módulos CSS** |
|  |  |  |  |
| **No hay espacio de nombres global** | \* | \* | ✔ |
| **estilos unidireccionales** | ✔ | ✔ | ✔ |
| **eliminación de código muerto** |  |  | ✔ |
| **minificación** |  |  | ✔ |
| **Las constantes se pueden compartir** |  |  |  |
| **resolución determinista** | \* | \* | ✔ |
| **Aislamiento** | \* | \* | ✔ |
| **extensible** | \* | \* | ✔ |
| **documentable** | N / A | N / A | N / A |
| **Presentable** | N / A | N / A | N / A |
|  |  |  |  |
| [ ✔ Cumplido] [\* Pseudo cumplido] |  |  |  |
|  |  |  |  |



Esto es una mejora importante! No sólo pudimos refactorizar toda nuestra aplicación con la mínima cantidad de cambios, pero también hemos sido capaces de cumplir muchos de nuestros criterios de U & I. Ahora estamos en condiciones de construir componentes predecibles, minify nuestras clases CSS y eliminar cualquier estilos que no se hace referencia en nuestra aplicación. ¡Asombroso!

En el siguiente capítulo, vamos a convertir todo lo que hemos aprendido en la cabeza mediante la exploración de los estilos en línea.

**Capítulo 8: Exploración de estilos en línea**

En el capítulo anterior hemos introducido módulos CSS y descubiertas algunas técnicas inteligentes para mitigar muchos temas de desarrollo de interfaz de usuario común. Hemos mejorado drásticamente el estado de nuestros componentes U & I, sin embargo, que técnicamente no hemos cumplido todo nuestro contrato.

Si esto es un requisito en su aplicación, a continuación, los estilos en línea puede ser la solución. estilos en línea? De Verdad? ¡Sí!

En noviembre de 2014, [Christopher Chedeau](https://twitter.com/Vjeux)45, De Facebook, dio una charla titulada [Reaccionar:](https://speakerdeck.com/vjeux/react-css-in-js) [CSS en JS](https://speakerdeck.com/vjeux/react-css-in-js)46. Se examinó siete problemas con la escritura de CSS a escala. Uso de la tarea de la construcción de un botón simple como un ejemplo, se dirigió a través de algunos enfoques bastante complejas para resolver los primeros cinco de ellos y dejando que los dos últimos sin resolver. Luego dejó caer una bomba bastante grande, declarando que todos los siete problemas podrían resolverse mediante el uso de estilos en línea definidas en el local de reaccionar componente.

A pesar de que afirma en sus diapositivas que su objetivo no era convencer a los desarrolladores a caer CSS y JS utilizar en su lugar, eso es exactamente lo que empezó a suceder. La charla estimuló una intensa actividad.

**¿Cuáles son los estilos en línea**

Si usted ha estado siguiendo a lo largo, que ya ha sido presentado a inline estilos. En resumen, es la metodología de aplicación de estilos directamente sobre el elemento con JavaScript. ¿Por qué puede ser esto útil? Aquí hay algunas razones rápidas que saldrán a la luz a medida que avanzamos a través de los siguientes capítulos:

• Cascade-menos: El miedo la naturaleza “global” del CSS es castrado.

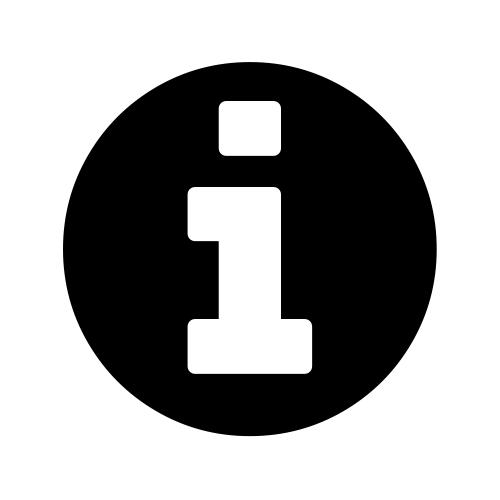
1. <https://twitter.com/Vjeux>
2. <https://speakerdeck.com/vjeux/react-css-in-js>

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 106 |

* **Todo JavaScript:** Una sensación que tengo es que algunas personas como y prefieren trabajar en todo JavaScript.
* **Estilos dinámicos**: “Estado” es en gran parte una preocupación JavaScript. Si quieres estilo / necesidad de cambiar basa en condiciones dinámicas (estados) en su sitio, puede tener sentido de manejar el estilo relacionado con el cambio de estado, junto con todo lo demás.

**Estilos en línea en Acción**

A diferencia de antes, donde teníamos que instalar las dependencias adicionales, ya tenemos los estilos en línea que nos ha proporcionado. Todo lo que se necesita es utilizar elestilo prop en los elementos apropiados.



Debido a la naturaleza modular de nuestros componentes, ha sido sin esfuerzo para cambiar entre estas diversas tecnologías. De hecho, puede hacer que los dos módulos CSS y estilos en línea trabajar juntos a través de una biblioteca de U & I.

**Configuración de los estilos**

Vamos a empezar con nuestra estilos directorio, ya que utilizamos estos a través de nuestra aplicación.

**src / estilos / theme.js**

* por defecto de exportación {
* $ APP: 'todos aplicación',

3

4$ MainBackgroundColor: '# F1F1F1',

5$ MainFontFamily: 'Helvetica Neue, Arial, sans-serif',

6

7$ LightGray: '#fafafa',

8$ Gris\_oscuro: '# 888888'

9 }

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 107 |

Esto debería resultar familiar; simplemente hemos convertido nuestro viejotheme.scss a su equivalente JS, donde las variables y sus valores son la llaves y valors en un objeto de JS. Seguimos a adherirse a la JSel caso de Carmel convenio de denominación.

Ya no hay necesidad main.scss, Por lo que eliminar el archivo y quitar su referencia dentro de index.js. Por el momento, vamos a ignorar globals.scss como vamos a aplicar estilos globales más adelante en el capítulo.

**Refactorización <app />**

Visita el Aplicación estilos:

**src / componentes / App / App.styles.js**

* por defecto de exportación {

2aplicación: {},

3todosList: {

4margen: '50px auto',

5anchoMax: 800

* },
* todosListItem: {
* margen: '10px 0'

9}

1. }

El tipo de archivo se actualiza a JS y es una buena práctica para indicar que el archivo es de un estilo tipo; por lo tanto, la ModuleName.styles.js. Los estilos se convierten en sus JS equivalentes y se exportan para el consumo.

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 108 |

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 estilos de importación de' ./App.styles';

7

8 clase App extiende Componente {

9

1. Constructor (... args) {
2. ...
3. }

13

1. componentDidMount () {
2. ...
3. }

17

1. toggleTodo = (id) => (e) => {
2. ...
3. };

21

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

25

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <Div style = {} styles.app>
5. <TodosList pStyles = {} estilos>
6. {\_.map (todos, this.generateTodosListItem)}
7. <TodosListInfo todos = {todos} />
8. </ TodosList>
9. </ Div>

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 109 |

1. );
2. }
3. }

38

1. Aplicación por defecto de exportación;

Similar a módulos CSS capítulo anterior, importamos nuestra estilos y hacer referencia a la llave en los elementos apropiados. Una diferencia importante aquí es que ya no estamos pasando por nombres CSS como accesorios, pero los estilos actuales, de ahí la razón por la que optó por la apuntalar interfaz para pasar por nuestros estilos.

**Refactorización <TodosList />**

Vamos a empezar con nuestros primeros estilos:

**src / componentes / TodosList / TodosList.styles.js**

* por defecto de exportación {

2todosList: {

3ListStyle: 'ninguno',

4padding: '10px 15px'

* }

6 }

Los estilos se convierten a su formato JS equivalente.

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 110 |

**src / componentes / TodosList / TodosList.js**

* // dependencias
* \_ importar de 'lodash';
* importación Reaccionar de 'reaccionar';
* PropTypes de importación de 'prop-tipos';

5

6 // dependencias locales

7 estilos de importación'./TodosList.styles;

8

9 const TodosList = {(niños, pStyles}) => (

1. <Ul style = {\_. Asignar ({}, styles.todosList, pStyles.todosList)}>
2. {niños}
3. </ Ul>
4. );

14

1. TodosList.defaultProps = {
2. niños: [],
3. pStyles: {
4. todosList: {}
5. }
6. };

21

1. TodosList.propTypes = {
2. hijos: PropTypes.oneOfType ([
3. PropTypes.array,
4. PropTypes.element
5. ]),
6. pStyles: PropTypes.shape ({
7. todosList: PropTypes.object
8. })
9. };

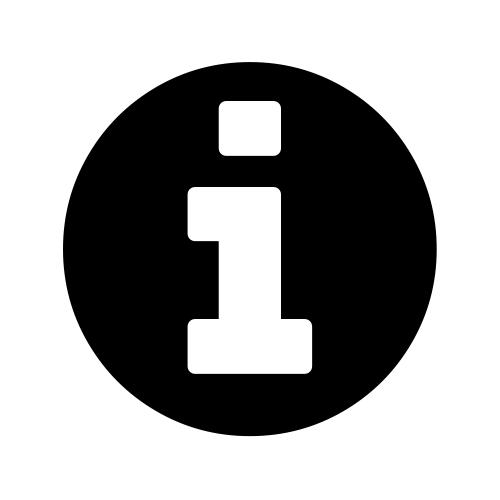
31

32 TodosList predeterminado de exportación;

Actualización de la dependencia, se fusionan los estilos internos y lateral y, por último, actualizar el puntal

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 111 |

definiciones.



Debe aplicar los estilos internos primero y luego pStyles, De modo que los estilos personalizados transmitidos a través de los apoyos tienen prioridad en la operación de fusión.

**Refactorización <TodosListInfo />**

En primer lugar, vamos a actualizar nuestro archivo de estilos:

**src / componentes / TodosListInfo / TodosListInfo.styles.js**

* tema de la importación de '../../styles/theme';

2

3 por defecto de exportación {

4todosListInfo: {

5Color:. $ gris\_oscuro tema,

6fontSize: 14,

7textAlign: 'derecho'

* }

9 }

Los estilos se convierten en sus equivalentes JS y que hacen referencia a nuestra temavariables 's, según sea necesario.

**src / componentes / TodosListInfo / TodosListInfo.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 estilos de importación'./TodosListInfo.styles;

7

8 const TodosListInfo = ({todos}) => {

9...

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 112 |

1. regreso (
2. <Li style = {} styles.todosListInfo>
3. {CompletedTodosCount} / {} todosCount completado
4. </ Li>
5. );
6. };

dieciséis

1. ...

importamos estilos y hacer referencia a las teclas correspondientes.

**Refactorización <TodosListItems />**

Vamos a empezar con nuestros estilos:

**src / componentes / TodosListItem / TodosListItem.styles.js**

* tema de la importación de '../../styles/theme';

2

3 por defecto de exportación {

4todosListItem: {

5Antecedentes:. $ lightGray tema,

6borderRadius: 5,

7frontera: '1px solid # E1E1E1',

8boxShadow: '0 2px 1px 0 RGBA (0, 0, 0, 0.2)',

9Color:. $ gris\_oscuro tema,

1. cursor: 'puntero',
2. fontSize: '2rem',
3. padding: '15px 20px',
4. posición: 'relativo',
5. transición: 'todo 0.2s facilidad',
6. // estilos en línea no son compatibles con CSS complejo pseudo-selectores,
7. // animaciones de cuadros clave, preguntas de los medios, etc.
8. // más adelante vamos a explorar cómo podemos conseguir este tipo de

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 113 |

1. // funcionalidad con bibliotecas de código abierto útiles
2. // Y: hover {
3. // opacidad: 0,8
4. //}
5. },

25

1. completedTodosListItem: {
2. Antecedentes:. $ lightGray tema,
3. boxShadow: 'inserción 0 1px 2px 0 RGBA (0, 0, 0, 0.3)',
4. Color: '#AAAAAA',
5. textDecoration: 'line-a través de',
6. 3 primeros
7. }
8. }

Hacemos los cambios necesarios para actualizar el archivo de nuestros estilos, pero hemos comentado algunas de las propiedades CSS más complejos, ya que no cuentan con los estilos en línea base. Nos encargaremos de hacer los estilos en línea totalmente funcional muy pronto.

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias
* \_ importar de 'lodash';
* PropTypes de importación de 'prop-tipos';

4 importación Reaccionar de 'reaccionar';

5

6 // dependencias locales

7 estilos de importación'./TodosListItem.styles;

8

9 const TodosListItem = ({pStyles, todo, handleClick}) => (

1. <li
2. style = {\_. asignar ({},
3. styles.todosListItem,
4. pStyles.todosListItem,
5. todo.completed && styles.completedTodosListItem,

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 114 |

1. todo.completed && pStyles.completedTodosListItem
2. )}
3. onclick = {handleClick (todo.id)}>
4. {} Todo.description
5. </ Li>
6. );

21

1. TodosListItem.defaultProps = {
2. que hacer: {},
3. toggleTodo: \_.noop,
4. pStyles: {
5. todosListItem: {},
6. completedTodosListItem: {}
7. }
8. };

30

1. TodosListItem.propTypes = {
2. handleClick: PropTypes.func,
3. TODO: PropTypes.shape ({
4. completado: PropTypes.boolean,
5. Descripción: PropTypes.string,
6. ID: PropTypes.number
7. }),
8. pStyles: PropTypes.shape ({
9. todosListItem: PropTypes.object,
10. completedTodosListItem: PropTypes.object
11. })
12. };

43

1. TodosListItem predeterminado de exportación;

Importamos nuestros estilos y aplicarlos adecuadamente a los elementos correctos. Continuamos siguiendo el mismo patrón para dar una mayor importancia a nuestrapStyles. Por último, nos aseguramos de actualizar nuestrodefaultProps y propTypes definiciones para reflejar nuestros cambios.

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 115 |

Si visita la aplicación, se encuentra que la aplicación está casi a la par con el lugar donde estábamos en nuestros capítulos anteriores:



Hay, sin embargo, algunas cosas a tener en cuenta:

* Ya no tenemos nuestros estilos CSS globales. Podríamos incluir un archivo CSS para aplicar estos estilos diferentes, sin embargo, vamos a optar por una mejor solución todo-en-JS.
* Con los estilos en línea, no tenemos muchas de las características de CSS avanzadas, como pseudo-selectores, las consultas multimedia, animaciones de fotogramas clave, etc. Sin embargo, hay algunas librerías muy útiles que se adelante la exploración de esta funcionalidad.
* Ya no tenemos la compone o se extiende funcionalidad, por lo que nos fusionamos nuestros dos objetos diferentes, ya sea con la ES6 ... operador de difusión o asignar/unir.

No está mal para algo que nos proporcionan fuera de la caja. Vaya por delante e inspeccionar los elementos DOM. Se puede ver todos los estilos aplicados directamente sobre los elementos HTML, como era de sospecha.

**Estilos en línea mejorada**

estilos en línea son muy ordenada, pero no se puede construir nada sustancial si no somos capaces de utilizar algunas de las características más avanzadas de CSS. Es por esto que se requiere una mejor solución

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 116 |

para ayudarnos a lograr una funcionalidad importante como los estilos globales, funciones avanzadas de CSS, etc.

**¿Cuál es el radio?**

Reunirse [Radio](http://formidable.com/open-source/radium/)47! Una biblioteca bien documentado y apoyado, hecho por[Formidable](https://formidable.com/open-source/) [laboratorios](https://formidable.com/open-source/)48, Para complementar los estilos en línea con toda la funcionalidad deseable que necesitamos. Ahí son otras bibliotecas por ahí, sin embargo, el radio es robusto con una comunidad activa, por lo que vamos a utilizar para demostrar los estilos en línea en toda su gloria.

**Radio en Acción**

En primer lugar, tendremos que instalar dos dependencias:

$ NPM i radio --save radio-normalizar

Estamos utilizando radio-normalizarpara ayudar a los estilos de reinicio del navegador. Esto es similar a lo que se ve en unareset.css o una normalize.css para lograr un comportamiento común a través de diferentes navegadores.

Una vez que se instalan estas dependencias, añadiendo el radio de nuestro proyecto es fácil. Vamos a utilizar algunas de las características más populares de Radium, pero recomendamos mirando a través de su documentación para familiarizarse con muchas de las otras características proporcionadas.

**Configuración de los estilos**

Actualizar el globales presentar estilos.

1. <http://formidable.com/open-source/radium/>
2. <https://formidable.com/open-source/>

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 117 |

**src / estilos / globals.styles.js**

* v importación de './theme';

2

3 por defecto de exportación {

4'\*, \*:antes después': {

5boxSizing: 'hereda'

* },
* html: {
* boxSizing: 'border-box'

9},

1. cuerpo: {
2. Antecedentes:. v $ mainBackgroundColor,
3. fontSize: 10,
4. fontFamily:. v $ mainFontFamily,
5. margin: 0,
6. padding: 0
7. }
8. }

Los estilos fueron convertidos a su equivalente JS. Estamos dispuestos a aplicar estos estilos globales.

**Refactorización <app />**

Ya que estamos aplicando todos nuestros principales estilos dentro de app.js, Vamos a seguir adelante y empezar con este archivo.

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 118 |

**src / componentes / App / app.js**

* // dependencias

2 ...

3 importación {estilo, StyleRoot} de 'radio';

4 importación de normalizar 'radio-normalizar';

5

6 // dependencias locales

7 ...

8 globalStyles importación de ../../styles/globals.styles';

9

1. clase App extiende Componente {
2. Constructor (... args) {
3. ...
4. }

15

1. componentDidMount () {
2. ...
3. }

19

1. toggleTodo = (id) => (e) => {
2. ...
3. };

23

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

27

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <StyleRoot>
5. <Las reglas de estilo = {} normalizar />
6. <Las reglas de estilo = {} globalStyles />
7. <Div style = {} styles.app>

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 119 |

1. <TodosList pStyles = {} estilos>

|  |  |
| --- | --- |
| 36 | {\_.map (todos, this.generateTodosListItem)} |
| 37 | <TodosListInfo todos = {todos} /> |

1. </ TodosList>
2. </ Div>
3. </ StyleRoot>
4. );
5. }
6. }

44

1. ...

importamos Estilo y StyleRoot de radio y por supuesto, radio-normalizar.

Estilo permite para nosotros para inyectar CSS en nuestra aplicación, lo que vamos a utilizar para nuestra radio-normalizar. los StyleRoot Módulo envuelve toda la aplicación como una forma de mantener la totalidad de su Estado en el Reaccionar contexto. Si no está familiarizado con la Reaccionar[contexto](https://facebook.github.io/react/docs/context.html)49, Es la forma como las bibliotecas reaccionar-redux son capaces de pasar datos desde reduxen su aplicación. usos Radiumcontexto para mantener el estado estilos.

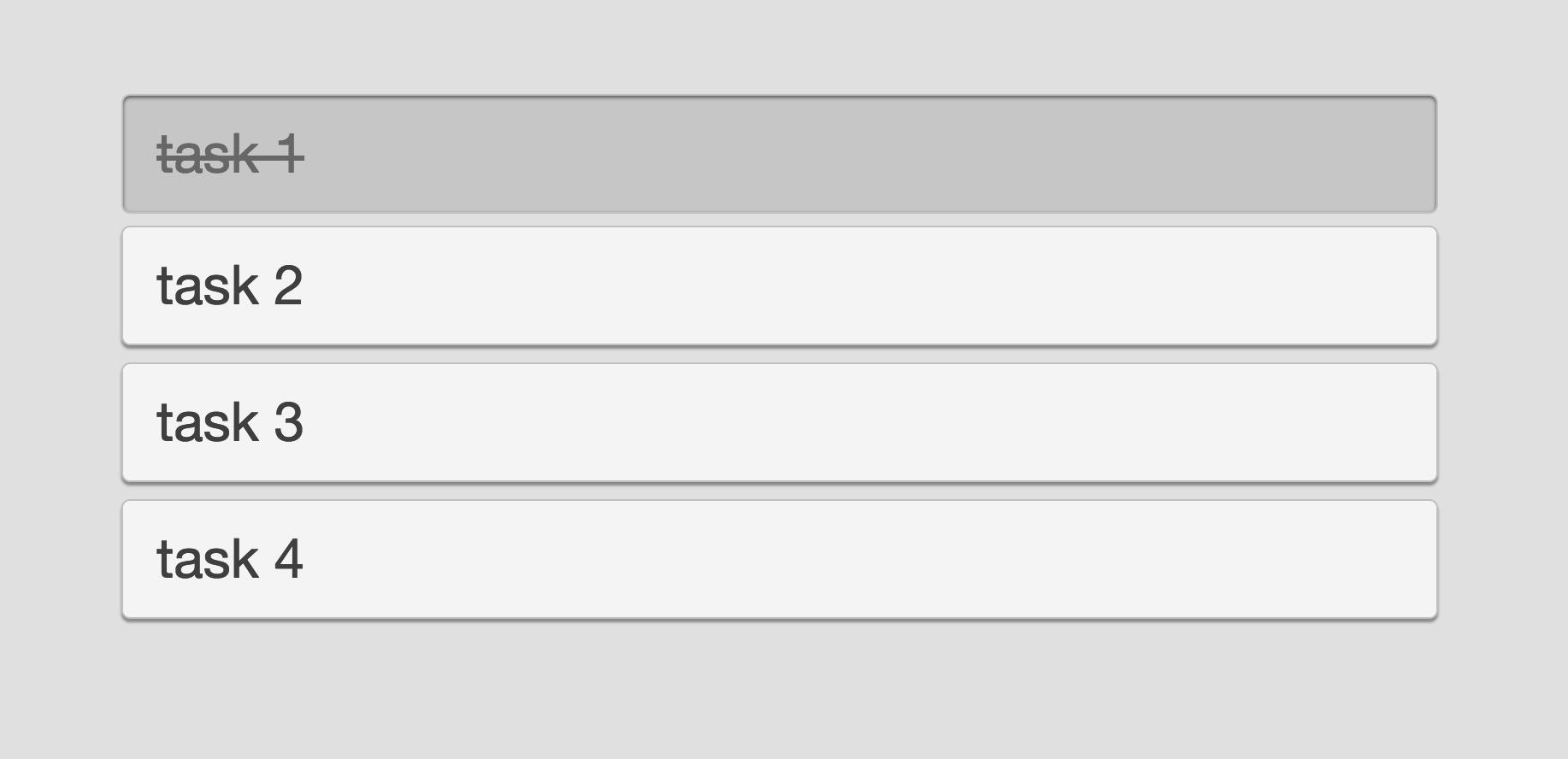
Hemos terminado toda nuestra aplicación con <StyleRoot> y aprobar normalizar a la primera instancia de <Style> componentes y nuestra gloablStylesa la segunda instancia. los<Style> componente es lo que permite que nosotros inyectamos CSS en nuestra aplicación y ya no estamos definiendo cualquier ámbito particular en sus apoyos, por defecto a ser global.

¡Eso es todo! Ahora tenemos Radium corriendo con nuestra aplicación y se está ya Suplemento-ing todos nuestros estilos en línea.

Ya se puede ver que algunos de los restablecimientos de los navegadores Web que se han aplicado.

1. <https://facebook.github.io/react/docs/context.html>

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 120 |



**Refactorización <TodosList />**

No estábamos usando cualquier característica CSS avanzados en este componente, por lo que nada para cambiar aparte de envolver con nuestro componente Radio.

**src / componentes / TodosList / TodosList.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 const TodosList = {(niños, pStyles}) => (

8<Style ul = {[styles.todosList, pStyles.todosList]}>

* {niños}

10</ Ul>

11 );

12

13 ...

14

15 defecto de exportación Radium (TodosList);

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 121 |

El estilo formaciónsintaxis es otra característica que proporciona Radium. En lugar defusionar / como signo, Podemos utilizar el formación sintaxis y tienen estilos aplicados en el orden en que están definida. Eso es limpio!

**Refactorización <TodosListInfo />**

Muy similar a la anterior:

**src / componentes / TodosListInfo / TodosListInfo.js**

* // dependencias

2 ...

3 importar el radio de 'radio';

4

5 // dependencias locales

6 ...

7

8 const TodosListInfo = ({todos}) => {

9...

1. };
2. ...
3. defecto de exportación Radium (TodosListInfo);

Somos importadores de radio y envuelto nuestro componente de manera que se ha mejorado con características de Radium.

**Refactorización <TodosListItems />**

Vamos a empezar con nuestros estilos:

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 122 |

**src / componentes / TodosListItem / TodosListItem.styles.js**

* tema de la importación de '../../styles/theme';

2

3 por defecto de exportación {

4todosListItem: {

5Antecedentes:. $ lightGray tema,

6borderRadius: 5,

7frontera: '1px solid # E1E1E1',

8boxShadow: '0 2px 1px 0 RGBA (0, 0, 0, 0.2)',

9Color:. $ gris\_oscuro tema,

1. cursor: 'puntero',
2. fontSize: '2rem',
3. padding: '15px 20px',
4. posición: 'relativo',
5. transición: 'todo 0.2s facilidad',
6. ': Hover': {
7. opacidad: 0,8
8. }
9. },

19

1. completedTodosListItem: {
2. Antecedentes:. $ lightGray tema,
3. boxShadow: 'inserción 0 1px 2px 0 RGBA (0, 0, 0, 0.3)',
4. Color: '#AAAAAA',
5. textDecoration: 'line-a través de',
6. 3 primeros
7. }
8. }

Nos Descomentar nuestro código y permitir :flotar definiéndola como una explícita cuerda en los estilos de objeto.

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 123 |

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias

2 ...

3 importar el radio de 'radio';

4

5 // dependencias locales

6 ...

7

8 const TodosListItem = ({pStyles, todo, handleClick}) => (

9<li

1. style = {[
2. styles.todosListItem,
3. pStyles.todosListItem,
4. todo.completed && styles.completedTodosListItem,
5. todo.completed && pStyles.completedTodosListItem
6. ]}
7. onclick = {handleClick (todo.id)}>
8. {} Todo.description
9. </ Li>
10. );

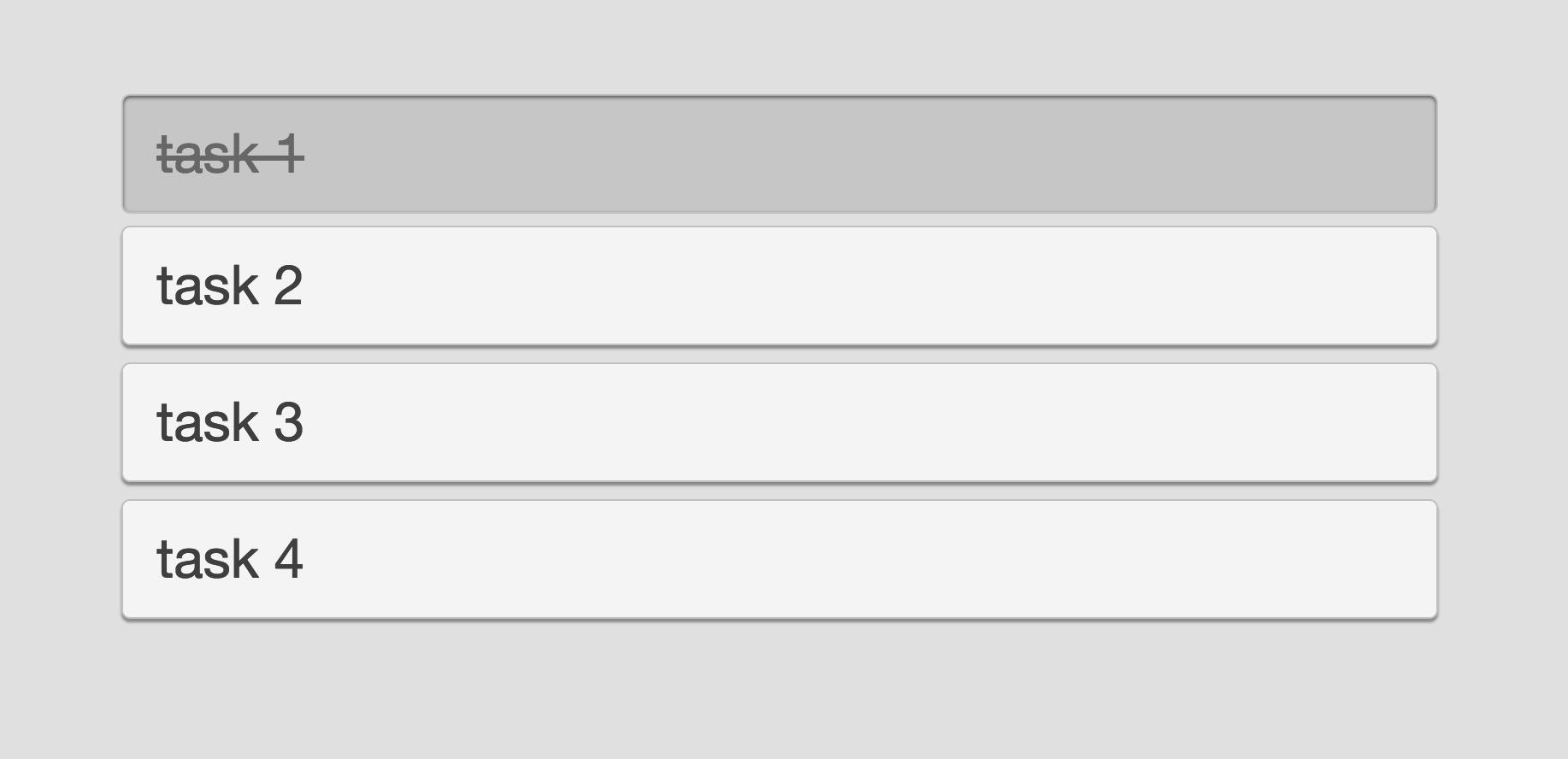
20

1. ...
2. defecto de exportación Radium (TodosListItem);

Necesitamos Radio y actualizar nuestra exportación, al igual que antes, y seguir utilizando la sintaxis de matrices de fusionar nuestros estilos.

Visita su navegador y usted encontrará que la aplicación tiene todas las características que cabría esperar. Todo esto se trabajó con unas pocas líneas de código adicionales. Esto es sólo el comienzo, ya te digo!

|  |  |
| --- | --- |
| Capítulo 8: Exploración de estilos en línea | 124 |



**Ejercicio sugerido**

Para entender mejor el radio y en línea estilos, recomiendo experimentar con lo siguiente:

* Añadir otros estilos globales para su aplicación. Por ejemplo, añadir un valor predeterminadocolor de fondo a todos div elementos dentro de la aplicación.
* Experimento con preguntas de los medios.
* Experimento con animaciones de cuadros clave CSS.
* preprocesadores CSS nos proporcionan muchas funciones útiles, como las manipulaciones de color. ¿Cómo hacer esto con JS? Es posible que desee ver en un módulo de NPM pequeña llamada[color](https://github.com/brehaut/color-js)50.
* ... y cualquier otra cosa que le gustaría, por supuesto.

**Resumen**

En este capítulo, convertimos toda nuestra aplicación para consumir los estilos en línea con algunos cambios menores. ¿Dónde nos encontramos en nuestra lista de comprobación T & I?

1. <https://github.com/brehaut/color-js>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Capítulo 8: Exploración de estilos en línea |  |  |  | 125 |
|  | **CSS** | **SCSS** | **Módulos CSS** | **estilos en línea** |
|  |  |  |  |  |
| **No hay espacio de nombres global** | \* | \* | ✔ | ✔ |
| **estilos unidireccionales** | ✔ | ✔ | ✔ | ✔ |
| **eliminación de código muerto** |  |  | ✔ | ✔ |
| **minificación** |  |  | ✔ | ✔ |
| **Las constantes se pueden compartir** |  |  |  | ✔ |
| **resolución determinista** | \* | \* | ✔ | ✔ |
| **Aislamiento** | \* | \* | ✔ | ✔ |
| **extensible** | \* | \* | ✔ | ✔ |
| **documentable** | N / A | N / A | N / A | N / A |
| **Presentable** | N / A | N / A | N / A | N / A |



[ ✔ Cumplido] [\* Pseudo cumplido]

Whoora! Finalmente hemos cumplido todas nuestras especificaciones U & I descritos enCapitulo 2, Excepto unos pocos que nos estamos dejando intencionadamente hasta los últimos capítulos. Con los estilos en línea, ya no tenemos que preocuparnos por las colisiones de nombres y gestión de la dependencia. Continuamos proporcionando una API intuitiva para los estilos personalizados y somos capaces de compartir código entre nuestros estilos y plantillas.

En el siguiente capítulo, vamos a ver cómo podemos aprovechar los estilos en línea para lograr capacidades en tiempo real. Qué significa eso? Usted pronto descubrirá ...

**Capítulo 9: La adición de capacidades de tiempo real**

En el capítulo anterior hemos descubierto cómo podemos utilizar un viejo paradigma, tales como los estilos en línea, en las aplicaciones modernas para cumplir con muchos de los criterios U & I descritos en Capítulo 2.

Así que, ¿qué vamos a hacer en este capítulo?

En este capítulo, vamos a tener estilos en línea a un nivel totalmente nuevo. Digamos que usted quiere construir una aplicación que podría intercambiar variables, temas y estilos sobre la marcha. ¿Cómo hacer que con CSS, preprocesadores CSS, CSS módulos, etc.? Es necesario que depender de algunas herramientas de construcción inteligentes para ayudarle a alcanzar capacidades seudo tiempo real. Sin embargo, desde que hemos duplicado-caídos en JS-todo, podemos aprovechar muchas de sus potentes cualidades para agregar comportamientos en tiempo real.

**¿Cuál es el tema Wrap?**

En este capítulo, se introduce [es el tema de envoltura](https://github.com/FarhadG/theme-wrap)51, Una biblioteca de código abierto que he construido para ayudar a gestionar los estilos en línea, proporcionar una gestión tema, ofrecer apoyo mezclar el estilo, y, en última instancia, capacidades en tiempo real.

Tu puedes pensar en es el tema de envolturacomo un papel de regalo para sus aplicaciones Reaccionar. Inteligente, lo sé! :)

**Wrap tema en Acción**

Lo primero es instalar la biblioteca:

1. <https://github.com/FarhadG/theme-wrap>

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 127 |

$ Es el tema de envoltura --save NPM i

Una vez instalado, podemos empezar a explorar toda su funcionalidad diferente.

**Refactorización <app />**

Vamos a empezar con la raíz de nuestros componentes:

**src / componentes / App / app.js**

* // dependencias

2 ...

3 {} la importación ThemeWrapProvider de 'tema-wrap';

4

5 // dependencias locales

6 ...

7 tema de la importación de '../../styles/theme';

8

9 clase App extiende Componente {

10

1. Constructor (... args) {
2. ...
3. }

14

1. componentDidMount () {
2. ...
3. }

18

1. toggleTodo = (id) => (e) => {
2. .
3. };

22

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

26

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 128 |

1. render () {
2. const {todos} = this.state;
3. regreso (
4. <StyleRoot>
5. <Las reglas de estilo = {} normalizar />
6. <Las reglas de estilo = {} globalStyles />
7. <ThemeWrapProvider tema = {tema}>
8. ...
9. </ ThemeWrapProvider>
10. </ StyleRoot>
11. );
12. }
13. }

40

1. ...

Hemos terminado con nuestra aplicación ThemeWrapProvider y pasar nuestra tema mediante el temaapuntalar. Ahora tenemostema en React de contexto.

**Refactorización <TodosList />**

Vamos a actualizar estilos:

**src / componentes / TodosList / TodosList.styles.js**

* por defecto de exportación () => ({

2todosList: {

3ListStyle: 'ninguno',

4padding: '10px 15px'

* }

6 });

Exportamos función dónde temase proporciona como primer argumento en nuestra función de estilos (no se utiliza aquí). Tenga en cuenta la declaración implícita de retorno ES6 donde estamos devolviendo un objeto con una sola tecla,todosList.

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 129 |

**src / componentes / TodosList / TodosList.js**

* // dependencias

2 ...

3 {} la importación applyThemeWrap de 'tema-wrap';

4

5 // dependencias locales

6 ...

7

8 TodosList const exportación = {(niños, pStyles, twStyles}) => (

9<Style ul = {[twStyles.todosList, pStyles.todosList]}>

1. {niños}
2. </ Ul>
3. );

13

1. ...
2. \_.flow predeterminado de exportación (
3. Radio,
4. applyThemeWrap (estilos)
5. ) (TodosList);

importamos applyThemeWrap de es el tema de envoltura para mejorar nuestro componente mediante la aplicación de nuestros estilos con el proporcionado tema. También necesitaremos una utilidad de la composición de funciones para ayudarnos a envolver nuestro componente con varias funciones de orden superior. Si ha utilizadoRedux, Entonces has probablemente más utilizada componer, Permitiendo que los componentes a ser envueltos con varias funciones de orden superior. Si usted no está familiarizado con este concepto, me gustaría recomendar altamente mirando a los muchos artículos que explican la función currificación, funciones de orden superior, etc. Es una actividad digna de entender sus beneficios y cómo utilizarlo, además de que es bastante Reaccionar frecuente en las aplicaciones. Dicho esto,fluir es la utilidad de lodash con un beneficio adicional que se aplica a los argumentos de la función en el orden que se leen, en lugar de componer siendo derecha a izquierda.

Pasamos estilos a applyThemeWrap por lo que tenemos tema proporcionado a nuestra estilospara el consumo. En segundo lugar, actualizamos las referencias a los estilos detwStyles (Abreviatura de estilos temáticos envuelta) se transmite a través de accesoriospor applyThemeWrap.

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 130 |

Por último, exportar nuestro componente base sin todas las mejoras, en los casos en los que queremos probar o hacer referencia a sus definiciones prop.

Vamos a hacer un resumen rápido de los cambios necesarios:

* nuestros estilos estilos Convertirse en un función con tema como primer argumento (si es necesario).
* componentes de envoltura con applyThemeWrap y pasar el componente de estilos como primer argumento.
* estilos de referencia de twStyles transmitido a través accesorios.
* Exportar el componente base.

**Refactorización <TodosListInfo />**

Vamos a empezar con los estilos:

**src / componentes / TodosListInfo / TodosListInfo.styles.js**

* por defecto de exportación (tema) => ({

2todosListInfo: {

3Color:. $ gris\_oscuro tema,

4fontSize: 14,

5textAlign: 'derecho'

* }

7 });

Nuestros estilos se convierten en una función con temasiempre como el primer argumento. Ya no necesitamos tener acceso a nuestrotema directamente.

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 131 |

**src / componentes / TodosListInfo / TodosListInfo.js**

* // dependencias

2 ...

3 {} la importación applyThemeWrap de 'tema-wrap';

4

5 // dependencias locales

6 ...

7 {} la importación de TodosListItem '../TodosListItem/TodosListItem';

8

9 const exportación TodosListInfo = ({todos, twStyles}) => {

1. ...
2. regreso (
3. <Li style = {} twStyles.todosListInfo>
4. {CompletedTodosCount} / {} todosCount completado
5. </ Li>
6. );
7. };

17

1. ...
2. \_.flow predeterminado de exportación (
3. Radio,
4. applyThemeWrap (estilos)
5. ) (TodosListInfo);

Envolver con el componente applyThemeWrap y pasar en nuestra estilos. Luego, a través de estilos de referenciatwStyles transmitido a través accesorios. No se olvide deexportar la base compo-nente y actualizar la forma de importar TodosListItem para referencia prop.

**Refactorización <TodosListItem />**

Al igual que antes, es más fácil comenzar con nuestra estilos:

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 132 |

**src / componentes / TodosListItem / TodosListItem.styles.js**

* por defecto de exportación (tema) => ({

2todosListItem: {

3Antecedentes:. $ lightGray tema,

4borderRadius: 5,

5frontera: '1px solid # E1E1E1',

6boxShadow: '0 2px 1px 0 RGBA (0, 0, 0, 0.2)',

7Color:. $ gris\_oscuro tema,

8cursor: 'puntero',

9fontSize: '2rem',

1. padding: '15px 20px',
2. posición: 'relativo',
3. transición: 'todo 0.2s facilidad',
4. ': Hover': {
5. opacidad: 0,8
6. }
7. },

17

1. completedTodosListItem: {
2. Antecedentes:. $ lightGray tema,
3. boxShadow: 'inserción 0 1px 2px 0 RGBA (0, 0, 0, 0.3)',
4. Color: '#AAAAAA',
5. textDecoration: 'line-a través de',
6. 3 primeros
7. }
8. });

Convertir estilos en una función y referencia tema, según sea necesario.

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 133 |

**src / componentes / TodosListItem / TodosListItem.js**

* // dependencias

2 ...

3 {} la importación applyThemeWrap de 'tema-wrap';

4

5 // dependencias locales

6 ...

7

8 const exportación TodosListItem = ({

9pStyles, todo, twStyles, handleClick

1. }) => (
2. <li
3. style = {[
4. twStyles.todosListItem,
5. pStyles.todosListItem,
6. todo.completed && twStyles.completedTodosListItem,
7. todo.completed && pStyles.completedTodosListItem
8. ]}
9. onclick = {handleClick (todo.id)}>
10. {} Todo.description
11. </ Li>
12. );

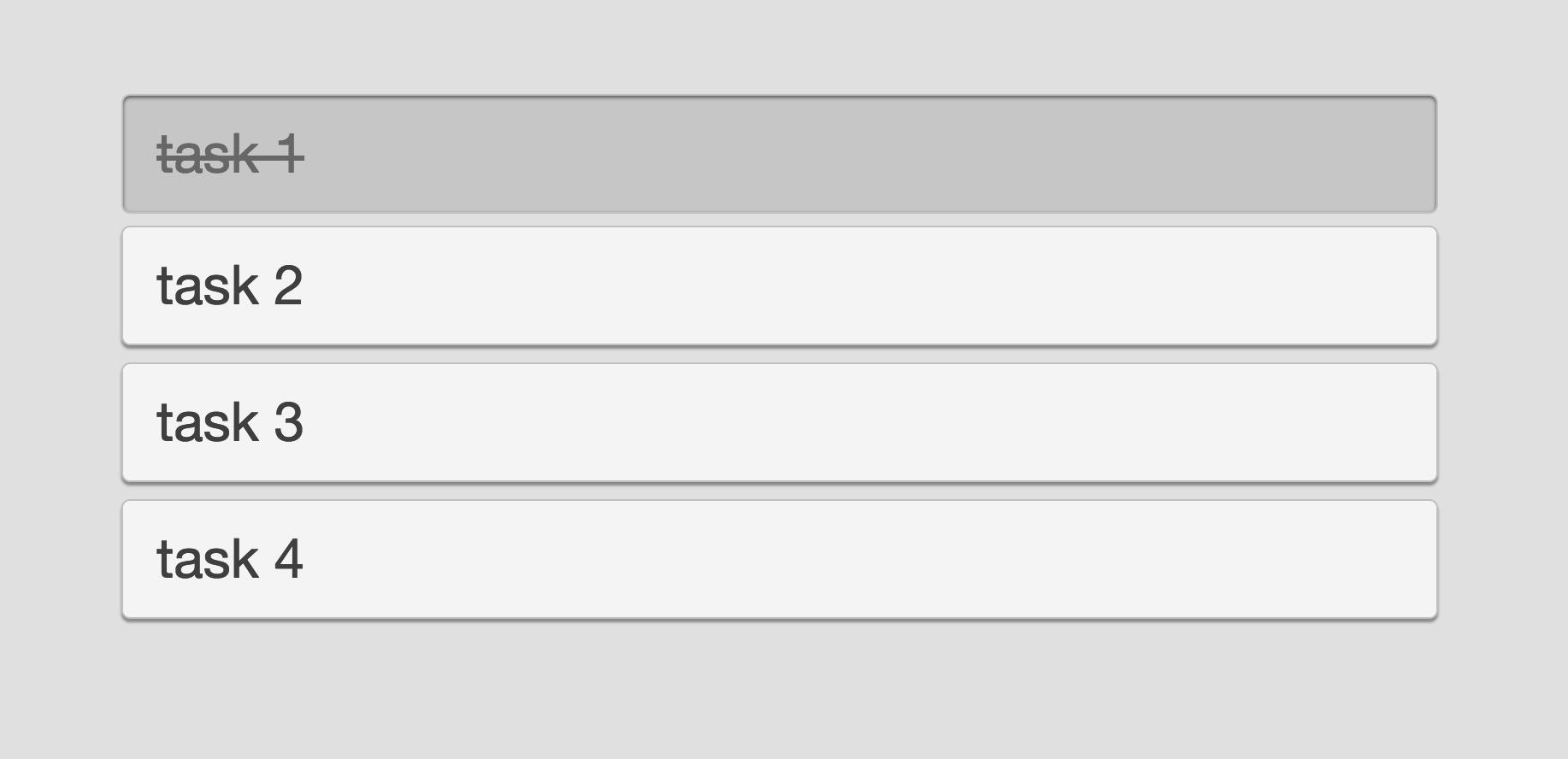
22

1. ...
2. \_.flow predeterminado de exportación (
3. Radio,
4. applyThemeWrap (estilos)
5. ) (TodosListItem);

¿Se esperaba algo diferente? Espero que no, porque pasamos por los mismos pasos como se mencionó antes.

Seguir adelante y visitar su navegador y asegurarse de que todo funciona:

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 134 |



¿Por qué hicimos todos estos cambios? Éstos son algunos de los beneficios a tener en cuenta, que vamos a apalancar la brevedad:

* ThemeWrapProvider Ahora gestiona nuestra tema. Es decir, ofrecemos nuestra tema una vez y todos nuestros componentes conseguir el acceso a esta única fuente de la verdad.
* ThemeWrapProvider gestiona nuestra tema con capacidades de tiempo real. Es decir, si actualizamos nuestras variables temáticos, todos nuestros componentes se vuelven conscientes de este cambio y reaccionar al instante. Esto puede ser útil si usted quiere construir una aplicación que permite a un estilo dinámico, la edición en vivo, intercambio de tema, etc.
* Ya que tema se trata como datos, Podemos seguir patrones similares a la forma redux y reaccionar-redux el tratamiento de los datos mediante la exposición de lo que es esencial para los componentes conectados.

Vamos a empezar a pasar un buen rato ...

**tema dinámico**

Vamos a tener un poco de diversión. Realmente para remachar el clavo, es necesario introducir unas cuantas más variables en nuestrotema:

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 135 |

**src / estilos / theme.js**

* por defecto de exportación {
* $ APP: 'todos aplicación',

3

4$ MainBackgroundColor: '# F1F1F1',

5$ MainFontFamily: 'Helvetica Neue, Arial, sans-serif',

6

7$ LightGray: '#fafafa',

8$ Gris\_oscuro: '# 888888',

9

1. $ PrimaryColor: 'rojo',
2. $ SecondaryColor: 'verde',
3. $ TertiaryColor: 'azul',
4. };

Hemos añadido tres nuevas variables en nuestro tema. No dude en ser creativo mediante la adición de cualquier paleta de colores desea:

**src / componentes / TodosList / TodosList.styles.js**

* por defecto de exportación (tema) => ({

2todosList: {

3ListStyle: 'ninguno',

4padding: '10px 15px',

5frontera: `10px sólida $ {.} secondaryColor tema $`,

6Antecedentes:. $ primaryColor tema,

7transición: 'todo 0.5s facilidad',

8borderRadius: 5

9}

1. });

Sí, esto no va a quedar muy bonito :)

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 136 |

**src / componentes / TodosListInfo / TodosListInfo.styles.js**

* por defecto de exportación (tema) => ({

2todosListInfo: {

3Color:. $ secondaryColor tema,

4fontSize: 14,

5textAlign: 'derecho'

* }

7 });

Uno mas…

**src / componentes / TodosListItem / TodosListItem.styles.js**

* por defecto de exportación (tema) => ({

2todosListItem: {

3Antecedentes: el tema.$ lightGray,

4borderRadius: 5,

5frontera: `sólido 5px *ps*tema.pstertiaryColor*}*`,

6boxShadow: '0 2px 1px 0 RGBA (0, 0, 0, 0.2)',

7tema de color.$ gris\_oscuro,

8cursor: 'puntero',

9fontSize: '2rem',

1. padding: '15px 20px',
2. posición: 'relativo',
3. transición: 'todo 0.2s facilidad',
4. ': Hover': {
5. opacidad: 0,8
6. }
7. },

17

1. completedTodosListItem: {
2. Antecedentes: el tema.$ primaryColor,
3. frontera: `sólido 5px *ps*tema.pscolor secundario*}*`,
4. boxShadow: 'inserción 0 1px 2px 0 RGBA (0, 0, 0, 0.3)',
5. Color: '#AAAAAA',

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 137 |

1. textDecoration: 'line-a través de',
2. 3 primeros
3. }
4. });

Visita su navegador y verá que hemos hecho oficialmente nuestra lista de todos peor pesadilla de un diseñador.



¡Oye! Usted no eligió este libro de consejos de diseño, por lo tomas con calma o se siente libre para construir una paleta de color apropiado de su elección. Una de mis favoritas es recursos[Adobe](https://color.adobe.com/) [Color](https://color.adobe.com/)52, Cuando necesito inspiración para elegir una paleta de colores.

Pasando ...

En la actualidad, cuando camiar nuestros elementos de tareas, animan a sus valores actualizados. Estos valores son dinámicos y se pueden cambiar sobre la marcha, así que vamos a seguir adelante y ver cómo funcionaría:

1. <https://color.adobe.com/>

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 138 |

**src / componentes / App / app.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 clase App extiende Componente {

8

9Constructor (... args) {

1. super (... args);
2. this.state = {
3. Todos: {},
4. dynamicTheme: el tema
5. };
6. }

dieciséis

1. componentDidMount () {
2. ...
3. }

20

1. randomizeTheme = () => {
2. const ColorPalette = [
3. 'Rojo', 'verde', 'azul', 'amarillo',
4. 'Naranja', 'púrpura', 'cian'
5. ];
6. const dynamicTheme = {
7. ... this.state.dynamicTheme,
8. $ PrimaryColor: \_.sample (ColorPalette),
9. $ SecondaryColor: \_.sample (ColorPalette),
10. $ TertiaryColor: \_.sample (ColorPalette)
11. };
12. this.setState ({dynamicTheme});
13. };

34

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 139 |

1. toggleTodo = (id) => (e) => {
2. ...
3. this.randomizeTheme ();
4. };

39

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

43

1. render () {
2. ...
3. }
4. }

48

1. ...

Dado que actualizamos nuestra tema, Comenzamos a seguirlo en el interior de nuestro estado como componente dynamicTheme. Pasamos a este tema a través del estado de nuestro componente a la ThemeWrap-Proveedor.



Es ideal para gestionar nuestra tema en el interior de un gestor de estado, al igual Redux, Como una única fuente de verdad.

Añadimos una nueva función, randomizeTheme, Que es llamada cada vez que toggleTodo. Esta aleatoriedad se extiende por todotema variables y actualizaciones $ primaryColor,

$ secondaryColor y $ tertiaryColor.

Vaya por delante y echa un vistazo a su navegador! Cada vez que se pulsa varias un TODO, nuestro componente de reaccionar a nuestro tema actualizada en tiempo real, sin ningún tipo de actualización. ¡Increíble!

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 140 |



¿Cómo es esto útil? Imaginar escenarios donde los usuarios pueden cambiar su perfil por fuentes cambiantes, tamaños de fuente, esquemas de color enteros aplicación, etc., y podemos tener la aplicación completa reaccionar en consecuencia, en tiempo real, sin ningún tipo de actualización. Estos son los tipos de aplicaciones que podemos llegar a esperar en el futuro.

Recomiendo parar aquí y experimentar con diferentes estilos y elementos. Puede ser divertido y gratificante!

**mixins**

Mixins son otra funcionalidad útil proporcionada por es el tema de envoltura. Estos mixins son similares a lo que se obtiene de mixins Sass, pero en JS.

Por ejemplo, si tenemos una lista de listas, tales como una TodosList, NamesList, Etc., que tenía estilos comunes, mixins proporcionar una solución elegante para la gestión de estos estilos comunes. Vamos a ver cómo se ve que:

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 141 |

**src / estilos / mixins.js**

* {} la importación ThemeWrapMixin de 'tema-wrap';

2

3 por defecto de exportación nueva ThemeWrapMixin ()

4 .set ( 'lista', (tema) => ({

5ListStyle: 'ninguno',

6padding: '10px 15px',

7borderRadius: 5

8 }));

Importar ThemeWrapMixin de es el tema de envolturay crear una nueva instancia. Puede haber situaciones en las que tiene muchos diferentes instancias mixin, por lo que mediante la definición de los temas y mixins individuales, tenemos una buena cantidad de flexibilidad. En muchos casos, una única instancia está muy bien.

Después de crear una instancia, simplemente conjunto nuestra mixin mediante la definición de un nombre único, por ejemplo, lista, Y una función que es acepta tema como primer argumento. Esto es lo mismo tema gestionado por ThemeWrapProvider. ¡Ordenado!

Como se puede ver, estos estilos se asemejan a nuestra TodosList.styles.js. Esto se debe a que queremos extraer estos y tenerlos comparten entre todos los componentes de la lista. Por último, exportamos nuestramixins ejemplo.

Como hacer mixins consumida por conseguir ThemeWrapProvider? De la misma manera que la que consumetema. Vamos a ver cómo nos gustaría hacer lo siguiente:

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6 mixins importación de ../../styles/mixins';

7

8 clase App extiende Componente {

9

1. Constructor (... args) {

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 142 |

1. ...
2. }

13

1. componentDidMount () {
2. ...
3. }

17

1. randomizeTheme = () => {
2. ...
3. };

21

1. toggleTodo = (id) => (e) => {
2. ...
3. };

25

1. generateTodosListItem = (TODO, id) => (
2. ...
3. );

29

1. render () {
2. const {dynamicTheme, todos} = this.state;
3. regreso (
4. <StyleRoot>
5. <Las reglas de estilo = {} normalizar />
6. <Las reglas de estilo = {} globalStyles />
7. <ThemeWrapProvider tema = {} dynamicTheme

37 mixins = {} mixins>

1. ...
2. </ ThemeWrapProvider>
3. </ StyleRoot>
4. );
5. }
6. }

44

45 Aplicación por defecto de exportación;

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 143 |

Importar mixins y pasarlo como un apoyo a ThemeWrapProvider. ¡Eso es!

Ahora, sólo tenemos que utilizar nuestros mixins en nuestra TodosList componente:

**src / componentes / TodosList / TodosList.styles.js**

* por defecto de exportación (tema, mixins) => ({

2todosList: {

* ... mixins.get ( 'lista'),
* frontera: `10px sólida $ {.} secondaryColor tema $`,

5Antecedentes:. $ primaryColor tema,

6transición: 'todo 0.5s facilidad',

* }

8 });

Nuestra estilos archivo está realmente proporciona dos valores: tema y mixins. Eliminamos cualquiera de los estilos comunes yobtener ellos llamando a la adecuada mixinsvalor. El operador de difusión se utiliza para fusionar los estilos juntos.

Vamos a probar otro ejemplo:

**src / estilos / mixins.js**

* {} la importación ThemeWrapMixin de 'tema-wrap';

2

3 por defecto de exportación nueva ThemeWrapMixin ()

4 .set ( 'lista', (tema) => ({

5ListStyle: 'ninguno',

6padding: '10px 15px',

7transición: 'todo 0.5s facilidad',

8borderRadius: 5

9 }))

10

1. .set ( 'listitem', (tema) => ({
2. Antecedentes:. $ lightGray tema,
3. borderRadius: 5,
4. boxShadow: '0 2px 1px 0 RGBA (0, 0, 0, 0.2)',
5. Color:. $ gris\_oscuro tema,

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 144 |

1. cursor: 'puntero',
2. fontSize: '2rem',
3. padding: '15px 20px',
4. posición: 'relativo',
5. transición: 'todo 0.2s facilidad',
6. ': Hover': {
7. opacidad: 0,8
8. }
9. }))

Nos cadena la segunda definición mixin en nuestra mixin instancia para común listitemestilos. Vamos a ponerlo a utilizar:

**src / componentes / TodosListItem / TodosListItem.styles.js**

* por defecto de exportación (tema, mixins) => ({

2todosListItem: {

* ... mixins.get ( 'listitem'),
* frontera: `$ {sólidos 5px. tema de` $ tertiaryColor},
* },

6

7completedTodosListItem: {

* ... mixins.get ( 'listitem'),
* Antecedentes:. $ primaryColor tema,

1. frontera: `$ {sólidos 5px. tema de` $ secondaryColor},
2. boxShadow: 'inserción 0 1px 2px 0 RGBA (0, 0, 0, 0.3)',
3. Color: '#AAAAAA',
4. textDecoration: 'line-a través de',
5. 3 primeros
6. }
7. });

Visita su navegador y ver la aplicación en acción:

|  |  |
| --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | 145 |



Hay mucha más funcionalidad contenida en es el tema de envoltura. Por ejemplo, puede proporcionar múltiplesThemeWrapProviders en su aplicación. Es decir, puede tener varios temas dentro de su aplicación por haber diferentes partes de su aplicación contenida bajo diferentesThemeWrapProviders. Incluso se podría nido de suThemeWrapProviders. ¿Por qué quieres múltiples temas en su aplicación? Imaginar casos en aplicaciones de gran tamaño, por ejemplo, un sitio de noticias, donde cada sección de la aplicación tiene diferentes temas para los mismos componentes.

**Resumen**

En este capítulo, tomamos nuestra aplicación para el siguiente nivel, proporcionando estilos vivo-recargables, funcionalidad mixins potente, encapsulamiento tema, y ​​más ...

Ya que sólo hemos añadido la funcionalidad de nuestros estilos en línea, usted no debe sorprenderse de que seguimos para satisfacer todas las especificaciones U & I como antes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Capítulo 9: La adición de capacidades de tiempo real | |  |  | 146 |
|  | **CSS** | **SCSS** | **Módulos CSS** | **estilos en línea** |
|  |  |  |  |  |
| **No hay espacio de nombres global** | \* | \* | ✔ | ✔ |
| **estilos unidireccionales** | ✔ | ✔ | ✔ | ✔ |
| **eliminación de código muerto** |  |  | ✔ | ✔ |
| **minificación** |  |  | ✔ | ✔ |
| **Las constantes se pueden compartir** |  |  |  | ✔ |
| **resolución determinista** | \* | \* | ✔ | ✔ |
| **Aislamiento** | \* | \* | ✔ | ✔ |
| **extensible** | \* | \* | ✔ | ✔ |
| **documentable** | ✔ | ✔ | ✔ | ✔ |
| **Presentable** | ✔ | ✔ | ✔ | ✔ |



[ ✔ Cumplido] [\* Pseudo cumplido]

Antes de continuar con los siguientes capítulos, experimentar con es el tema de envolturay empezar a pensar en diferentes formas de apalancamiento este tipo de funcionalidad. Una cosa a destacar es que todo esto es posible debido a la potencia y flexibilidad de los estilos en línea.

**Capítulo 10: Exposición**

En el capítulo anterior nos abrieron algunas capacidades de gran alcance, ofreciendo soporte para temas en tiempo real a nuestra aplicación. A pesar de que sólo tenemos un puñado de componentes, está empezando a ser compleja en la comprensión de las diferentes APIs que proporcionan. Este problema se vuelve cada vez más evidente en las bibliotecas más grandes U & I con cientos de componentes. ¿Cómo podemos, como desarrolladores, comunicamos las interfaces de nuestros componentes a otros desarrolladores, diseñadores, etc.?

Hay un pequeño secreto que he estado esperando a cabo en todo el libro! ¡Lo siento! Fue por su propio bien!

¿No tiene sentido construir y documentar sus componentes U & I en completo aislamiento? ¡Eso creo! No sólo esta estrategia proporciona la capacidad de crear, revisar y prueba de forma aislada, sino que también proporciona el contexto apropiado para asegurar que estamos construyendo es exactamente lo que se necesita - nada más y nada menos. Reunirse[Libro de cuentos](https://storybook.js.org/)53!

**¿Cuál es Storybook?**

Libro de cuentos, como se describe en su sitio, es el “entorno de desarrollo de interfaz de usuario Te encantará a utilizar.” Que captura mi experiencia de libro de cuentos. Es una herramienta muy útil que proporciona un entorno para construir, documentar y mostrar los componentes de forma aislada. En lugar de leer el código fuente de un componente para tratar de averiguar las distintas API proporcionadas, podemos visitar Libro de cuentos, donde se obtiene el componente y toda su documentación en un entorno sencillo e intuitivo.

**Libro de cuentos reaccionar en Acción**

Añadiendo Reaccionar Libro de cuentos para su aplicación es tan fácil como algunos comandos simples:

En primer lugar, se necesita el comando global instalada:

1. <https://storybook.js.org/>

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 148 |

$ NPM i -g getstorybook

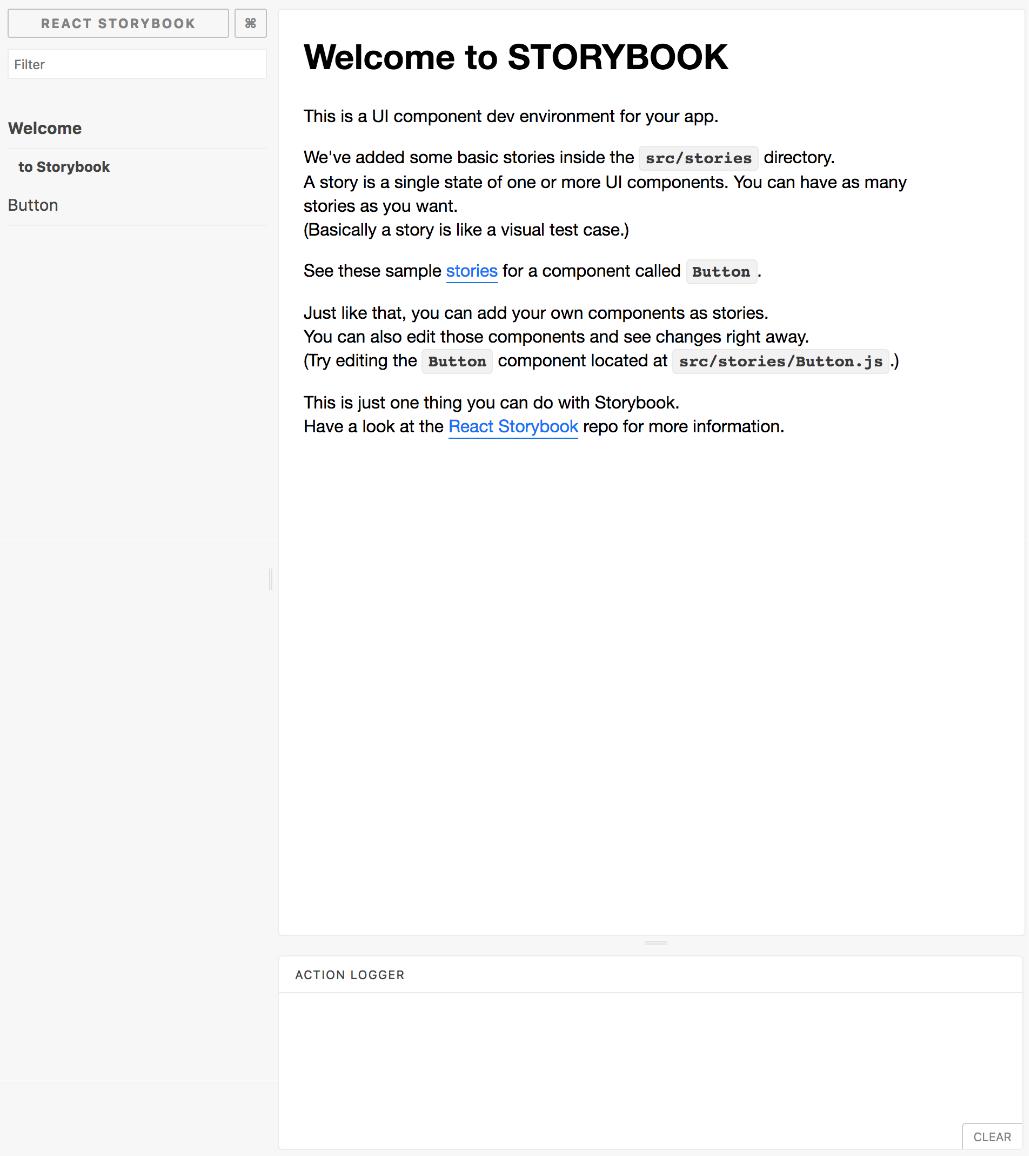
A continuación, en la raíz de nuestro directorio, ejecutamos el comando siguiente para Storybook configurar nuestra aplicación.

$ getstorybook

Una vez hecho esto, verá una .storybookdirectorio en la raíz de su proyecto. La belleza sobre Storybook, similar aCreate-reaccionar aplicación, Es que funciona fuera de la caja con algunas funciones maravillosa.

$ NPM plazo de cuentos

Si la ejecuta el comando siguiente, verá Libro de cuentos en la acción por la visita http: // localhost: 6006 o lo que sea la dirección que se muestra.



**IU libro de cuentos**

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 149 |

**Documentar <TodosList />**

Vamos a crear nuestra primera historia! En el mismo espíritu de mantener nuestros archivos de componentes co-localizados, vamos a albergar el archivo de historia en el mismo directorio.

**src / componentes / TodosList / TodosList.story.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* {} la importación de storiesOf '@ Kadira / libro de cuentos';

4

5 // dependencias locales

6 TodosList importación de './TodosList';

7

8 por defecto de exportación storiesOf ( 'TodosList', módulo)

9 .add ( 'vista por defecto', () => (

1. <TodosList>
2. </ TodosList>
3. ));

importamos storiesOf de libro de cuentosser capaz de crear una nueva historia. Añadimos una historia llamadavista predeterminada en el que no se representan nada.

Ahora, tenemos que importar esta historia en nuestro archivo de configuración del libro de cuentos:

**.storybook / config.js**

* // dependencias
* {} la importación de configuración de '@ Kadira / libro de cuentos';

3

4 loadStories function () {

* require ( '../ src / componentes / TodosList / TodosList.story');

6 }

7

8 Configurar (loadStories, módulo);

Si visitas el libro de cuentos de interfaz de usuario, debería ver un error:

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 150 |

* Error: mixins.get no es una función

Puedes adivinar qué es así?

Nuestro componente se basa en mixins y ya que estamos construyendo nuestro componente en forma aislada y sin es el tema de envoltura, Obtenemos que el error de romper. Vamos a ver cómo podemos resolver este:

**src / componentes / TodosList / TodosList.story.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* {} la importación de storiesOf '@ Kadira / libro de cuentos';

4 importación {estilo, StyleRoot} de 'radio';

5 importación de normalizar 'radio-normalizar';

6 {} la importación ThemeWrapProvider de 'tema-wrap';

7

8 // dependencias locales

9 TodosList importación de './TodosList';

1. globalStyles importación de ../../styles/globals.styles';
2. mixins importación de ../../styles/mixins';
3. tema de la importación de '../../styles/theme';

13

1. por defecto de exportación storiesOf ( 'TodosList', módulo)
2. .add ( 'vista por defecto', () => (
3. <StyleRoot>
4. <Las reglas de estilo = {} normalizar />
5. <Las reglas de estilo = {} globalStyles />
6. <Tema ThemeWrapProvider = {tema}

20 mixins = {} mixins>

1. <TodosList>
2. </ TodosList>
3. </ ThemeWrapProvider>
4. </ StyleRoot>
5. ));

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 151 |

Replicamos la misma funcionalidad en app.jsenvolviendo nuestro componente con todas las dependencias necesarias. Vaya por delante y echa un vistazo a la interfaz de usuario de cuentos.

No es muy emocionante, ya que no estamos mostrando todos los niños, por lo que dejó de documento que la funcionalidad:

**src / componentes / TodosList / TodosList.story.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 por defecto de exportación storiesOf ( 'TodosList', módulo)

8 .add ( 'vista por defecto', () => (

9...

1. ))
2. .add ( 'hace que los niños', () => (
3. <StyleRoot>
4. <Las reglas de estilo = {} normalizar />
5. <Las reglas de estilo = {} globalStyles />
6. <Tema ThemeWrapProvider = {tema}

17 mixins = {} mixins>

1. <TodosList>
2. {\_.map ([1, 2, 3], (i) => <li> {i} - Item </ li>)}
3. </ TodosList>
4. </ ThemeWrapProvider>
5. </ StyleRoot>
6. ))

Visita Storybook interfaz de usuario y debería ver una segunda historia, etiquetado como hace que los niños, Donde mostrar nuestro componente que contiene 3 li elementos.

Hay una buena cantidad de código repetido! ¿Cómo resolvemos esto? Podemos crear otro componente o el uso de Storybookdecorador:

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 152 |

**src / componentes / TodosList / TodosList.story.js**

* // dependencias

2 ...

3 importación {addDecorator, storiesOf} de '@ Kadira / libro de cuentos';

4

5 // dependencias locales

6 ...

7

8 const = Raíz (historia) => (

9<StyleRoot>

1. <Las reglas de estilo = {} normalizar />
2. <Las reglas de estilo = {} globalStyles />
3. <Tema ThemeWrapProvider = {tema}

13 mixins = {} mixins>

1. {historia()}
2. </ ThemeWrapProvider>
3. </ StyleRoot>
4. );

18

1. addDecorator (raíz);
2. // función auxiliar que genera un conjunto de niños
3. const generateChildren = (num = 5) => \_ (num)
4. .distancia()
5. .map ((i) => <clave li = {i}> {i} - Item </ li>)
6. .valor();
7. por defecto de exportación storiesOf ( 'TodosList', módulo)
8. .add ( 'vista por defecto', () => (
9. <TodosList>
10. </ TodosList>
11. ))
12. .add ( 'hace que los niños', () => (

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 153 |

1. <TodosList>
2. {GenerateChildren (5)}
3. </ TodosList>
4. ))

¡Muy agradable! Creamos unaRaíz componente que se historia vía accesorios y utilizarlo para decorar todas nuestras historias con addDecorator. Continuemos:

**src / componentes / TodosList / TodosList.story.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 por defecto de exportación storiesOf ( 'TodosList', módulo)

8 .add ( 'vista por defecto', () => ...)

9

1. .add ( 'hace que los niños', () => (...)
2. .add ( 'acepta estilos personalizados', () => {
3. const customStyles = {
4. todosList: {
5. fondo: 'naranja',
6. frontera: '10px sólido de color púrpura',
7. color morado',
8. fontSize: 40
9. }
10. };
11. regreso (
12. <TodosList pStyles = {customStyles}>
13. {GenerateChildren (10)}
14. </ TodosList>
15. );
16. });

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 154 |

Documentamos que TodosList acepta los estilos personalizados, que se pueden ver en la interfaz de usuario de cuentos.

Esto es muy bueno, pero ¿cómo saber a otros cómo utilizar estos componentes sin necesidad de mirar a su código fuente? Storybook ofrece muchos útilescomplementos¡para desarrolladores! Vamos a usar uno que hace exactamente lo que necesitamos:

$ NPM i @ libro de cuentos / addon-info --save-dev

Necesitamos actualizar nuestro archivo de configuración para utilizar este complemento:

**.storybook / config**

* // dependencias
* importación {configurar, setAddon} de '@ Kadira / libro de cuentos';

3 importación de infoAddon '@ libro de cuentos / addon-info';

4

5 setAddon (infoAddon);

6

7 loadStories function () {

* require ( '../ src / componentes / TodosList / TodosList.story');

9 }

10

1. Configurar (loadStories, módulo);

Importar infoAddon y utilizarlo con setAddon.



Si se produce un error que reaccionar-addons-crear-fragmento es indefinido, Puede instalarlo con el siguiente comando: NPM reacciono-complementos-create-fragmento --save.

Si queremos historias de componentes para incluir su documentación de la API, es necesario actualizar el añadir método para addWithInfo:

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 155 |

**src / componentes / TodosList / TodosList.story.js**

* // dependencias

2 ...

3

4 // dependencias locales

5 ...

6

7 por defecto de exportación storiesOf ( 'TodosList', módulo)

8 .addWithInfo ( 'vista por defecto', () => ...)

9

1. .addWithInfo ( 'hace que los niños', () => (...)
2. .addWithInfo ( 'acepta estilos personalizados', () => {...});

Visita Libro de cuentos y debería ver un pequeño icono, donde se puede alternar información y la interfaz de la com-ponente. Hay muchas otras opciones con este complemento, por lo que recomiendo experimentar para ver lo que funciona para usted.

Antes de pasar a nuestro siguiente componente, vamos a refactorizar nuestro archivo de configuración para que podamos Decorar todas nuestras historias con es el tema de envoltura y radio:

**.storybook / config.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* importación de normalizar 'radio-normalizar';

4 importación {estilo, StyleRoot} de 'radio';

5 importación de infoAddon '@ libro de cuentos / addon-info';

6 {} la importación ThemeWrapProvider de 'tema-wrap';

7 importación {addDecorator, configurar setAddon} de '@ Kadira / libro de cuentos \

8 ';

9

1. // dependencias locales
2. mixins importación de ../src/styles/mixins';
3. tema de la importación de '../src/styles/theme';

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 156 |

1. globalStyles importación de ../src/styles/globals.styles';
2. const = Raíz historia => (
3. <StyleRoot>
4. <Las reglas de estilo = {} normalizar />
5. <Las reglas de estilo = {} globalStyles />
6. <Tema ThemeWrapProvider = {tema}

20 mixins = {} mixins>

1. {historia()}
2. </ ThemeWrapProvider>
3. </ StyleRoot>
4. );

25

1. // adjuntar nuestro decorador
2. addDecorator (raíz);

28

1. // permitir componente info
2. setAddon (infoAddon);

31

1. configurar (() => {
2. require ( '../ src / componentes / TodosList / TodosList.story');
3. }, Módulo);

A continuación, actualice TodosList.story.js:

**src / componentes / TodosList / TodosList.story.js**

* // dependencias
* \_ importar de 'lodash';
* importación Reaccionar de 'reaccionar';
* {} la importación de storiesOf '@ Kadira / libro de cuentos';

5

6 // dependencias locales

7 TodosList importación de './TodosList';

8

9 // función auxiliar que genera un conjunto de niños

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 157 |

1. const generateChildren = (num = 5) => \_ (num)
2. .distancia()
3. .map ((i) => <li clave = {i}> Artículo # {i} </ li>)
4. .valor();

14

1. por defecto de exportación storiesOf ( 'TodosList', módulo)
2. .addWithInfo ( 'vista por defecto', () => (
3. <TodosList>
4. <Li> por defecto </ li>
5. </ TodosList>
6. ))

21

1. .addWithInfo ( 'hace que los niños', () => (
2. <TodosList>
3. {GenerateChildren (5)}
4. </ TodosList>
5. ))

27

1. .addWithInfo ( 'acepta estilos personalizados', () => {
2. const customStyles = {
3. todosList: {
4. fondo: 'naranja',
5. frontera: '10px sólido de color púrpura',
6. color morado',
7. fontSize: 40
8. }
9. };
10. regreso (
11. <TodosList pStyles = {customStyles}>
12. {GenerateChildren (10)}
13. </ TodosList>
14. );
15. });

Visita Storybook interfaz de usuario y todo debería funcionar como se espera.

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 158 |

**Documentar <TodosListInfo />**

Vamos a crear nuestra segunda historia de TodosListInfo.

En primer lugar actualizar el archivo de configuración de cuentos:

**.storybook / config.js**

* ...

2

3 configurar (() => {

* require ( '../ src / componentes / TodosList / TodosList.story');
* require ( '../ src / componentes / TodosListInfo / TodosListInfo.story'); 6 }, Módulo);

Tenemos que exigir que cada historia que se mostrará en el interior de la interfaz de usuario de cuentos.

Ahora, en nuestra historia:

**src / componentes / TodosListInfo / TodosListInfo.story.js**

* // dependencias
* \_ importar de 'lodash';
* importación Reaccionar de 'reaccionar';
* {} la importación de storiesOf '@ Kadira / libro de cuentos';

5

6 // dependencias locales

7 importación de TodosListInfo './TodosListInfo';

8

9 generateTodos const = (num = 5) => {

1. todos const = {};
2. \_.times (num, id => {
3. todos [id] = {
4. carné de identidad,
5. completado: Math.random ()> 0,5,
6. Descripción: `Tarea # $ {id}`
7. };
8. });

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 159 |

1. Todos retorno;
2. };

20

1. por defecto de exportación storiesOf ( 'TodosListInfo', módulo)
2. .addWithInfo ( 'vista por defecto', () => (
3. <TodosListInfo todos = {generateTodos ()} />
4. ));

¡Suficientemente simple! Desde que hemos encerrados estilo de este componente con el mundo exterior, que es casi imposible de peinar / moverlo, como se ve en el libro de cuentos de interfaz de usuario. No dude en ampliar la funcionalidad de este componente, si es necesario.

**Documentar <TodosListItem />**

Vamos a documentar nuestro último componente restante, TodosListItem.

**.storybook / config.js**

* ...

2

3 configurar (() => {

* require ( '../ src / componentes / TodosList / TodosList.story');
* require ( '../ src / componentes / TodosListInfo / TodosListInfo.story');
* require ( '../ src / componentes / TodosListItem / TodosListItem.story'); 7 }, Módulo);

En las piezas de la diversión:

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 160 |

**src / componentes / TodosListItem / TodosListItem.story.js**

* // dependencias
* importación Reaccionar de 'reaccionar';
* {} la importación de storiesOf '@ Kadira / libro de cuentos';

4

5 // dependencias locales

6 TodosListItem importación de './TodosListItem';

7

8 const sampleTodo = {

9id: 1,

1. completado: falso,
2. Descripción: 'Sample Todo'
3. };

13

1. const sampleCompletedTodo = {
2. id: 2,
3. completado: true,
4. Descripción: 'Sample Completado Todo'
5. };

19

1. por defecto de exportación storiesOf ( 'TodosListItem', módulo)
2. .addWithInfo ( 'vista por defecto', () => (
3. <TodosListItem />
4. ))

24

1. .addWithInfo ( 'acepta TODO a través de apoyos', () => (
2. <TodosListItem TODO = {sampleTodo} />
3. ))

28

1. .addWithInfo ( 'acepta completado TODO a través de apoyos', () => (
2. <TodosListItem TODO = {sampleCompletedTodo} />
3. ))

32

1. .addWithInfo ( 'acepta estilos personalizados a través de apoyos', () => {
2. const customStyles = {

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 161 |

1. todosListItem: {
2. frontera: '20px precipitó naranja',
3. Color naranja',
4. fontSize: 40,
5. margen: '0 auto',
6. padding: 10,
7. anchura: 500
8. }
9. };
10. regreso (
11. <TodosListItem TODO = {} sampleTodo

46 pStyles = {} customStyles />

1. );
2. })
3. .addWithInfo ( 'acepta estilos a través de apoyos completado personalizada', () => {
4. const customStyles = {
5. completado: {
6. backgroundColor: 'negro',
7. Color naranja'
8. }
9. };
10. regreso (
11. <TodosListItem TODO = {} sampleCompletedTodo

59 pStyles = {} customStyles />

1. );
2. });

¡Nada fuera de lo común! Creamos un par de historias, que describe las diversas API proporcionadas porTodosListItem. Sin embargo, tenemos una funcionalidad que no ha sido documentado:handleClick. Storybook ofrece otra característica útil,acción, Que ayuda a documentar nuestros controladores de eventos:

|  |  |
| --- | --- |
| Capítulo 10: Exposición | 162 |

**src / componentes / TodosListItem / TodosListItem.story.js**

* // dependencias

2 ...

3 importación {acción} storiesOf de '@ Kadira / libro de cuentos';

4

5 ...

6

7 .addWithInfo ( 'acepta devolución de llamada toggleTodo través de apoyos', () => (

8<TodosListItem TODO = {} sampleTodo

9 handleClick = {(ID) => (e) => acción ( 'alternar todo') (i \

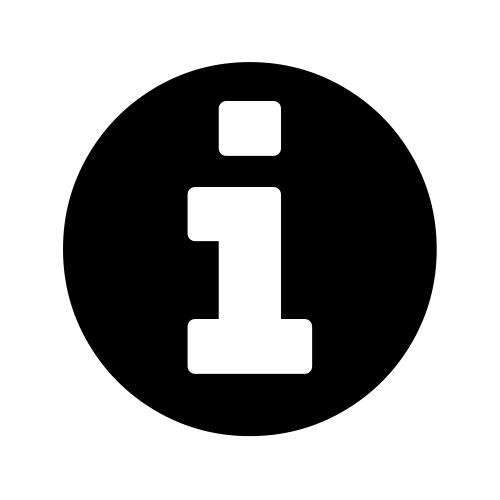
1. d)} />
2. ));

Dado que optamos por funciones de orden superior para manejar los controladores de eventos, nuestra controladores de eventos handleClick imita lo que teníamos originalmente en app.js. Si ahora clic en el elemento, verá un mensaje en elregistrador de la acción sección del Storybook UI.

**Resumen**

En este capítulo, hemos introducido Libro de cuentos en nuestra cadena de herramientas. Ahora debe tener la base para poder experimentar y crear otros addons de cuentos, decoradores y configuraciones. Libro de cuentos es a la vez útil y poderosa en proporcionar un entorno para crear rápidamente, documentar y mostrar los componentes.

Por último, podemos cumplir con los dos últimos criterios U & I que ignoramos en los capítulos anteriores.



Documentación y presentación de una biblioteca T & I no necesariamente depender de cualquier estrategia de estilo en particular, por lo que podríamos haber utilizado Storybook todo el tiempo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Capítulo 10: Exposición |  |  |  | 163 |
|  | **CSS** | **SCSS** | **Módulos CSS** | **estilos en línea** |
|  |  |  |  |  |
| **No hay espacio de nombres global** | \* | \* | ✔ | ✔ |
| **estilos unidireccionales** | ✔ | ✔ | ✔ | ✔ |
| **eliminación de código muerto** |  |  | ✔ | ✔ |
| **minificación** |  |  | ✔ | ✔ |
| **Las constantes se pueden compartir** |  |  |  | ✔ |
| **resolución determinista** | \* | \* | ✔ | ✔ |
| **Aislamiento** | \* | \* | ✔ | ✔ |
| **extensible** | \* | \* | ✔ | ✔ |
| **documentable** | ✔ | ✔ | ✔ | ✔ |
| **Presentable** | ✔ | ✔ | ✔ | ✔ |



[ ✔ Cumplido] [\* Pseudo cumplido]

Antes de pasar recomiendo crear otros componentes mediante Libro de cuentos y experimentar con otros complementos.

En el siguiente capítulo, vamos a cubrir otras tecnologías prometedoras vuelta de la esquina.

**Capítulo 11: El anticipar**

A lo largo de este libro, hemos cubierto varios potentes estrategias de U & I. ¿Hay otras estrategias no se incluyen en este libro? ¿Cómo se ve el panorama actual como?

Hay muchos, muchos más!

**exploraciones**

Hemos cubierto muchas estrategias diferentes a lo largo de este libro, pero el ecosistema para la construcción de la interfaz de usuario escalable es nuevo y continúa evolucionando rápidamente. Aquí hay algunas tecnologías para explorar:

**CSS Siguiente**

[CSS Siguiente](http://cssnext.io/)54 permite para que usted escriba “sintaxis CSS de mañana, hoy.” CSS Siguiente Ofertas muchas de las mismas características de un tradicional preprocesadores CSS, junto con algunos potentes capacidades adicionales. Por ejemplo, las variables, en el CSS A continuación, se pueden cambiar a través de JavaScript? WHAAaaaAATT? ¡Está bien! Podemos cambiar nuestras variables CSS en JavaScript. Esta[ejemplo](https://codepen.io/wesbos/pen/adQjoY)55, por [Wes Bos](https://twitter.com/wesbos)56, Hace un gran trabajo que muestra esta característica.

Esta es una característica interesante, porque podemos compartir constantes entre JavaScript y CSS. Estoy deseando CSS Siguiente siendo ampliamente apoyado para que pueda despreciar

es el tema de envoltura :)

**CSS en JS**

Escribir su CSS en JavaScript parece ser la estrategia más prometedora y que sigue siendo uno de los más rápidos en evolución entre las estrategias de estilo. En lugar de

1. <http://cssnext.io/>
2. <https://codepen.io/wesbos/pen/adQjoY>
3. <https://twitter.com/wesbos>

|  |  |
| --- | --- |
| Capítulo 11: El anticipar | 165 |

que detalle todas las diferentes implementaciones de CSS en JS, les recomiendo investigar en este informativo [repo](https://github.com/MicheleBertoli/css-in-js)57.

**Hardware Accelerated IU**

Por último, no necesariamente necesita atar todos nuestros componentes de DOM. Podríamos aprovecharlonay, mejor aún, aprovechar la GPU con WebGL. Si usted está buscando para construir una interfaz altamente interactivo y de inmersión, es posible que desee ver en estas bibliotecas:[Un cuadro](https://aframe.io/)58, [Tres reaccionar Procesador](https://github.com/toxicFork/react-three-renderer)59, Etc.

**Conclusión**

Gracias por tomarse el tiempo para ir a través de este libro. Espero que lo ha sido a la vez informativo y útil para la próxima vez que usted decide construir una aplicación. Por último, hemos cubierto muchas estrategias en este libro, pero no hemos tenido la oportunidad de dar un paso atrás y determinar sus aplicaciones prácticas. Como regla general, cuando me acerco a proyectos de interfaz de usuario, le pido a la siguiente serie de preguntas:

* Son los otros desarrolladores de front-end en el equipo sobre todo cómodo con CSS o un preprocesador de CSS?
* ¿Qué tan grande es la aplicación web?
* Es importante SEO? Si es así, no exprimir cada pedacito de la mejora necesaria?
* ¿La aplicación tiene que ser altamente interactivo y personalizable?
* ¿Servimos a diferentes aplicaciones para diferentes interfaces o estamos confiando en una sola aplicación que responde?

Sea cual sea la estrategia que decida utilizar, ya sea Sass, BEM, CSS módulos, los estilos en línea, CSS en JS, etc, no hay sustituto para una arquitectura bien definida.

Si tienes alguna pregunta o comentario, por favor refiérase a la [repositorio](https://github.com/FarhadG/ui-react)60 o ponerse en contacto conmigo directamente.

¡Gracias!

1. <https://github.com/MicheleBertoli/css-in-js>
2. <https://aframe.io/>
3. <https://github.com/toxicFork/react-three-renderer>
4. <https://github.com/FarhadG/ui-react>