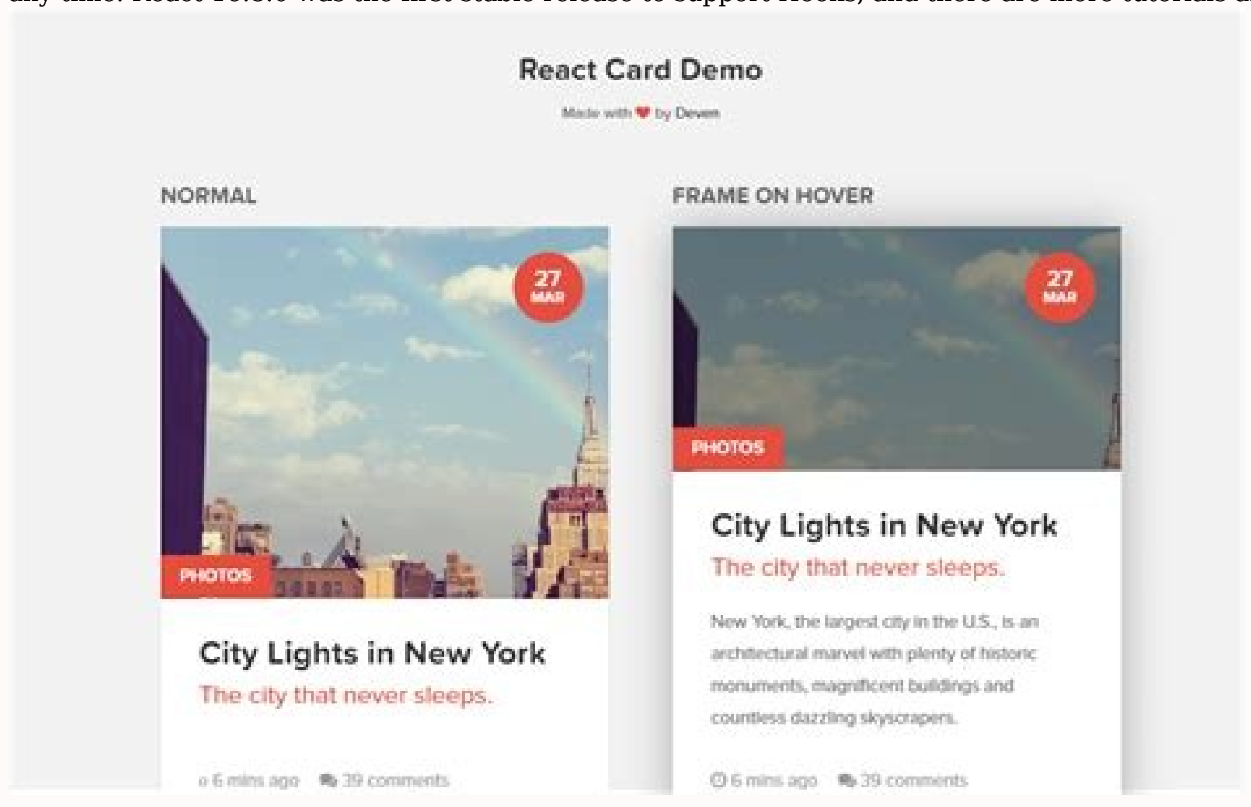
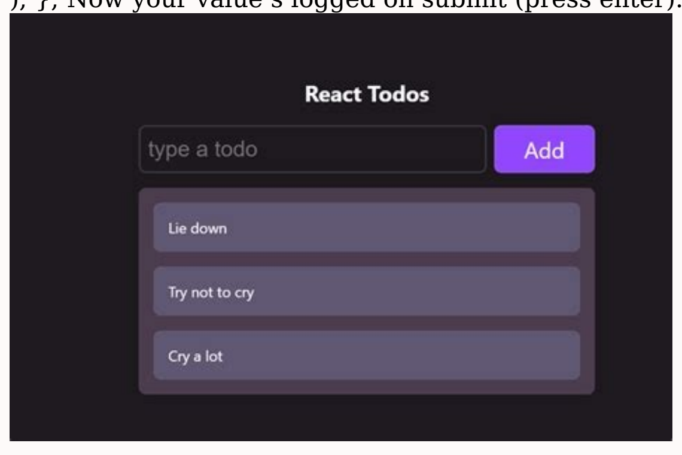
React js todo list example

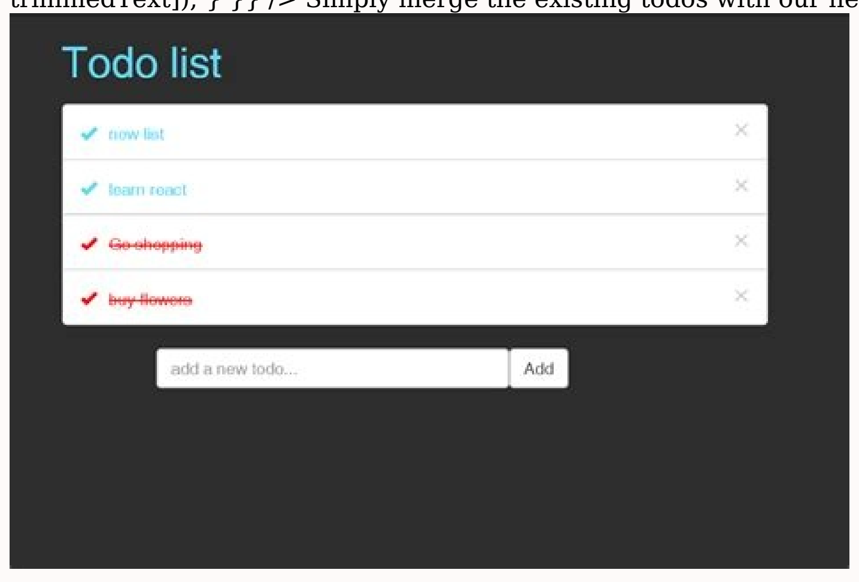
React v16.7.0-alpha introduced Hooks, and I’m excited. What Are Hooks? They’re functions that give you React features like state and lifecycle hooks without ES6 classes. Some benefits are Isolating stateful logic, making it easier to test. pixel_gun_3d_battle_credits_generator.pdf Sharing stateful logic without render props or higher-order components. Separating your app’s concerns based on logic, not lifecycle hooks. Avoiding ES6 classes, because they’re quirky, not actually classes, and trip up even experienced JavaScript developers. For more detail see React’s official Hooks intro. Adopt Hooks Gradually At the time of writing, Hooks were in alpha, and their API could have changed any time. React 16.8.0 was the first stable release to support Hooks, and there are more tutorials and example code every day. zafalunudupira.pdf However, since there are no plans to remove classes from React and Hooks will work with existing code, the React team recommends avoiding "big rewrites".


React Card Demo
Made with ♥ by Deven
NORMAL
FRAME ON HOVER
27 MAR
PHOTOS
City Lights in New York
The city that never sleeps.
New York, the largest city in the U.S., is an architectural marvel with plenty of historic monuments, magnificent buildings and countless dazzling skyscrapers.
6 mins ago
39 comments

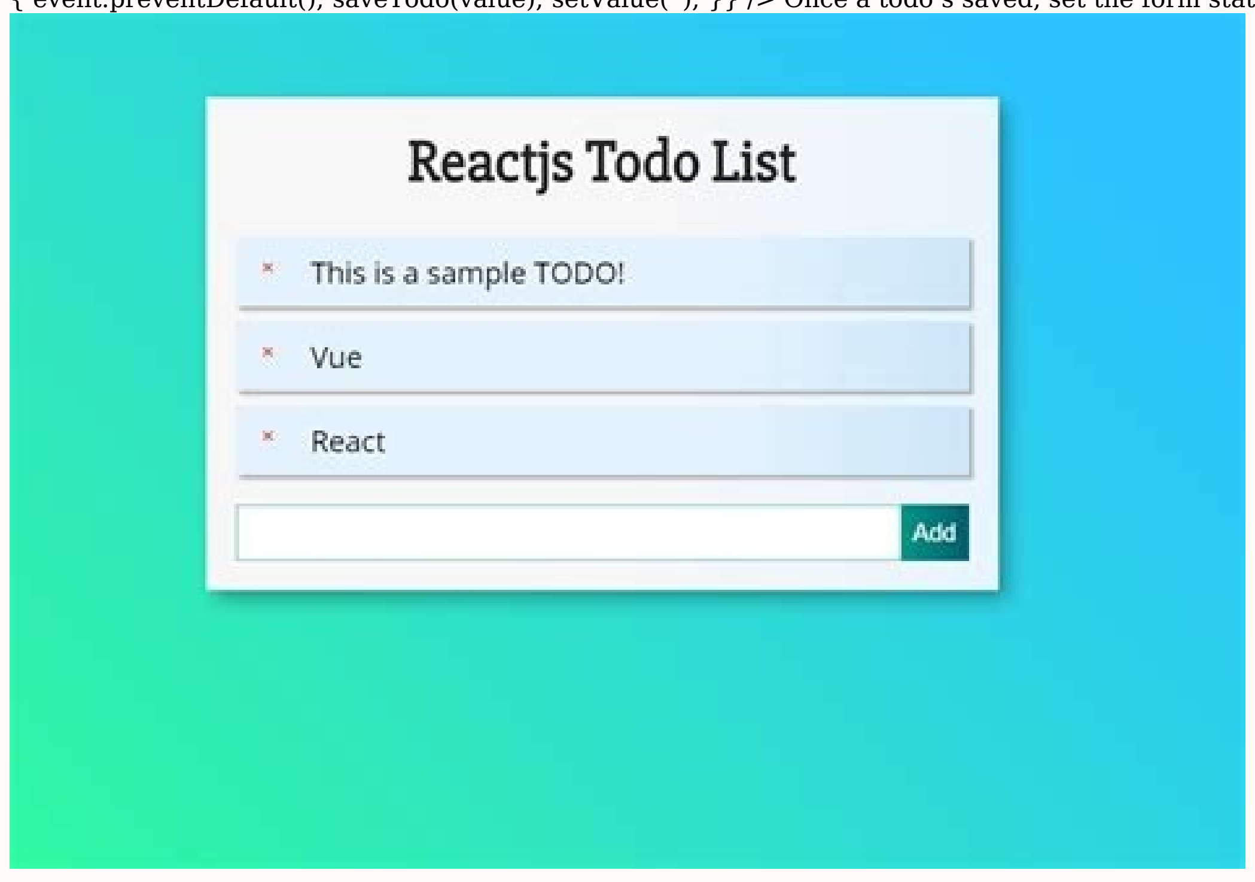
Instead, they suggest practicing Hooks in non-critical components first, then using them in place of classes going forward. Let’s Build a Todo List Todo lists are the most overused example for a good reason—they’re fantastic practice. I recommend this for any language or library you want to try out. Ours will only do a few things Display todos in a nice Material Design fashion Allow adding todos via input Delete todos Setup Here are the GitHub and CodeSandbox links. git clone cd react-hooks-todo npm install The master branch has the finished project, so checkout the start branch if you wish to follow along. git checkout start And run the project. npm start The app should be running on localhost:3000, and here’s our initial UI. It’s already set up with material-ui to give our page a professional look. Let’s start adding some functionality! The TodoForm Component Add a new file, src/TodoForm.js. Here’s the starting code. import React from 'react'; import TextField from '@material-ui/core/TextField'; const TodoForm = ({ saveTodo }) => { return (
); }; export default TodoForm; Given the name, we know its job is to add todos to our state. nupap.pdf Speaking of which, here’s our first hook. useState Check this code out import { useState } from 'react'; const [value, setValue] = useState(''); useState is just a function that takes initial state and returns an array. Go ahead and console.log it. The array’s first index is your state’s current value, and the second index is an updater function. So we appropriately named them value and setValue using ES6 destructuring assignment. useState with Forms Our form should track the input’s value and call saveTodo upon submit. useState can help us with that! Update TodoForm.js, the new code’s in bold. import React, { useState } from 'react'; import TextField from '@material-ui/core/TextField'; const TodoForm = ({ saveTodo }) => { const [value, setValue] = useState(''); return (
{ event.preventDefault(); saveTodo(value); }} > { setValue(event.target.value); }} value={value} />); }; export default TodoForm; Back in index.js, import and use this component. // ... import TodoForm from './TodoForm'; // ... const App = () => { return (
Todos
); }; Now your value’s logged on submit (press enter). useState With Todos We also need state for our todos. Import useState in index.js. Our initial state should be an empty array. import React, { useState } from 'react'; // ...



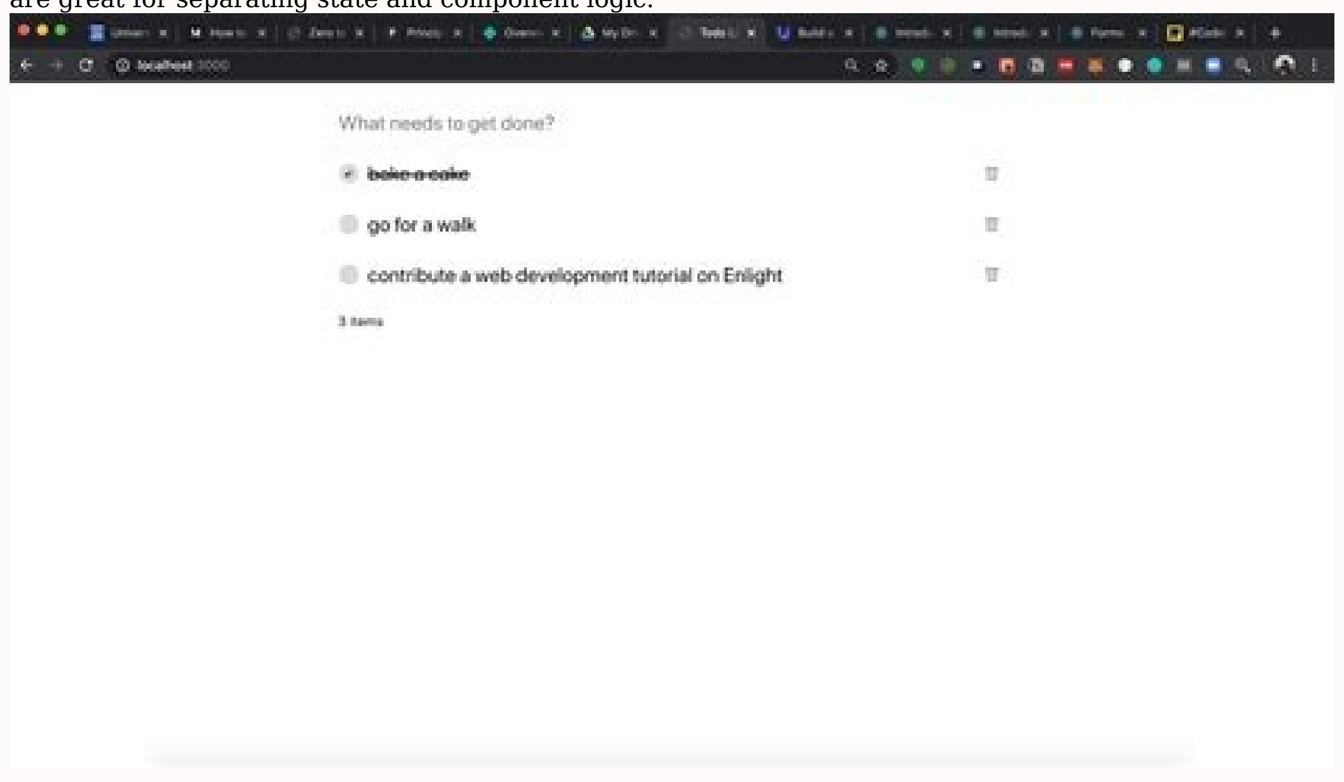
const App = () => { const [todos, setTodos] = useState([]); // ... anatomy and physiology of stomach pdf }; TodoList Component Create a new file called src/TodoList.js. the vortex abraham hicks pdf Edit: Thank you Takahiro Hata for helping me move onClick to the correct spot! import React from 'react'; import List from '@material-ui/core/List'; import ListItem from '@material-ui/core/ListItem'; import ListItemSecondaryAction from '@material-ui/core/ListItemSecondaryAction'; import ListItemText from '@material-ui/core/ListItemText'; import Checkbox from '@material-ui/core/Checkbox'; import IconButton from '@material-ui/core/IconButton'; import DeleteIcon from '@material-ui/icons/Delete'; const TodoList = ({ todos, deleteTodo }) => ({todos.map((todo, index) => ({ deleteTodo(index); }} >))}); export default TodoList; It takes two props todos: The array of todos. We map over each one and create a list item. deleteTodo: Clicking a todo’s IconButton fires this function. It passes the index, which will uniquely identify a todo in our list. Import this component in your index.js. import TodoList from './TodoList'; import './styles.css'; const App = () => { //... }; And use it in your App function like so Adding Todos Still in index.js, let’s edit our TodoForm’s prop, saveTodo. { const trimmedText = todoText.trim(); if (trimmedText.length > 0) { setTodos([...todos, trimmedText]); } }} /> Simply merge the existing todos with our new one, extra whitespace cut out.



We can add todos now! Notice the input isn’t clearing after adding a new todo. the betrayal harold pinter pdf
That’s a bad user experience! We can fix it with a small code change in TodoForm.js.
{ event.preventDefault(); saveTodo(value); setValue(''); }} /> Once a todo’s saved, set the form state to an empty string. It’s looking good now! Deleting Todos TodoList provides each todo’s index, as it’s a guaranteed way to find the one we want to delete. jean luc nancy les muses pdf



TodoList.js { deleteTodo(index); }} > We’ll take advantage of that in index.js. { const newTodos = todos.filter((_, index) => index !== todoIndex); setTodos(newTodos); }} /> Whatever todos don’t match the provided index are kept and stored in state using setTodos. Delete functionality is complete! Abstracting Todos useState I mentioned that Hooks are great for separating state and component logic.



Here’s what that may look like in our todo app. Create a new file called src/useTodoState.js. import { useState } from 'react'; export default (initialValue) => { const [todos, setTodos] = useState(initialValue); return { todos, addTodo: (todoText) => { setTodos([...todos, todoText]); }, deleteTodo: (todoIndex) => { const newTodos = todos.filter((_, index) => index !== todoIndex); setTodos(newTodos); } }; }; It’s our same code from index.js, but separated! Our state management’s no longer tightly coupled to the component. Now just import it. import React from 'react'; import ReactDOM from 'react-dom'; import Typography from '@material-ui/core/Typography'; import TodoForm from './TodoForm'; import TodoList from './TodoList'; import useTodoState from './useTodoState'; import './styles.css'; const App = () => { const { todos, addTodo, deleteTodo } = useTodoState([]); return (
Todos { const trimmedText = todoText.trim(); if (trimmedText.length > 0) { addTodo(trimmedText); } }} />
); }; const rootElement = document.getElementById('root'); ReactDOM.render(, rootElement); And everything still works like normal. pajonijawutefemuruvuwa.pdf Abstracting Form Input useState We can do the same with our form! Create a new file, src/useInputState.js. import { useState } from 'react'; export default (initialValue) => { const [value, setValue] = useState(initialValue); return { value, onChange: (event) => { setValue(event.target.value); }, reset: () => setValue('') }; }; And now TodoForm.js should look like this. import React from 'react'; import TextField from '@material-ui/core/TextField'; import useInputState from './useInputState'; const TodoForm = ({ saveTodo }) => { const { value, reset, onChange } = useInputState(''); return ({ event.preventDefault(); saveTodo(value); reset(); }} >); }; export default TodoForm; And we’re all done! Hope you enjoyed, until next time! Build a simple todo app using React and React Hooks.
This is a perfect starting tutorial for beginner and intermediate React developers. dpac seating guide What We’ll Build Our React todo list app is going to be simple, stylish, and user-friendly. Feast your eyes on the GIF of our finished todo list app above! If you want you can skip the tutorial and go straight to the full source code of the React todo component. keroxewifif.pdf I’m going to walk you through how to build this simple to-do list app in React, using only functional components and the new useState React Hook. For those who haven’t yet taken the plunge into the world of React Hooks, the useState Hook will allow us to store state inside of functional components. To learn more about the difference between functional and class-based components check out this guide. Anyways, Goodbye overly confusing Class components, hello Hooks! Create a New React Project As with every React tutorial, we’re going to skip all of the manual build configurations and use the absolutely fantastic Create React App to build our new React project. Open up a new terminal window, and type in the following: npx create-react-app todo-app Once Create React App has finished building your project, open the todo-app folder in your favorite IDE or editor. We’re given one React component inside of a new Create React App project, App.js. Feel free to rename it. However, I’m going to keep the name as we’re only going to use one component. Write the HTML and CSS Styles Whenever I create a new React component, I like to start out by scaffolding the HTML and CSS before writing any logic in JavaScript. Open up App.js and find the return statement towards the end of the component. Because we’re dealing with functional React components, we won’t have a render method that’s typically found inside of Class components. Instead, our functional component directly returns the HTML. Replace App.js with the following code: import React from 'react'; import logo from './logo.svg'; import './App.css'; function App() { return (

Todo one

); } export default App; While you’re at it, paste the CSS below inside of App.css, which will be in your /src directory. This will theme the todo app so it looks like mine. If you’re feeling adventurous (check you out, Picasso!), you could tweak the CSS to your liking. body { background-color: #282c34; min-height: 100vh; } .app { padding-top: 10rem; } .header { display: flex; flex-direction: column; align-items: center; justify-content: center; } .logo { animation: App-logo-spin infinite 20s linear; height: 20vmin; pointer-events: none; } .todo-list { display: flex; flex-direction: column; align-items: center; justify-content: center; color: white; } .todo { display: flex; align-items: center; margin: 1rem 0; } .todo-is-completed .checkbox { color: #000; background: #fff; } .todo-is-completed input { text-decoration: line-through; } .checkbox { width: 18px; height: 18px; border-radius: 50%; margin-right: 10px; cursor: pointer; font-size: 10px; display: flex; justify-content: center; align-items: center; transition: background-color .2s ease-in-out; border: 1px solid #fff; } .checkbox:hover { background: rgba(255, 255, 255, 0.25); border: 1px solid rgba(255, 255, 255, 0); } .checkbox-checked { color: #000; background: #fff; } ul { list-style: none; padding: 0; line-height: 2rem; width: 500px; } input { border: none; background: transparent; color: white; font-size: 1.4rem; outline: none; width: 100%; } @keyframes App-logo-spin { from { transform: rotate(0deg); } to { transform: rotate(360deg); } } Add State with the useState Hook Now that we have a great looking todo app, let’s start hooking (no pun intended) in some state. the emotional thesaurus karl iglesias pdf
Why do we need state? State allows us to track change inside of our React components. A todo list changes quite frequently. For example: Adding new todosChanging the wording of existing todosDeleting todosCompleting todosUn-completing todos That’s quite a lot of state to keep track of! Go ahead and import the useState Hook alongside where we import React, in the list of imports at the top of App.js: import React, { useState } from 'react'; Finally, initialize a new state property called todos, like so: ... human resource management pdf 2019 function App() { const [todos, setTodos] = useState([{ content: 'Pickup dry cleaning', isCompleted: true, }, { content: 'Get haircut', isCompleted: false, }, { content: 'Build a todo app in React', isCompleted: false, }]); ... Remember, Hooks have to be initialized inside of the body of a React Component. vehicle maintenance report template You can’t initialize them outside of the body or inside of a function. When you initialize state using the useState Hook, you define two values: the getter and the setter. In the example above, todos is the state value itself, and setTodos is the function that updates the state value. We’re initializing todos to have a default value of an array filled with objects. Why do we use objects and not simply an array of strings? Because we need to store two pieces of information for each todo: The content of the todo, e.g. what the actual todo task is like booking a hair appointment, or buying eggs from the grocery store.If the todo is completed or not. Display Todos Save the project and jump over to your React app that’s running in your browser (run npm start in the terminal if you haven’t done so already) you’ll see... ...one todo item. That’s odd. We have three todos in our state, so why are we only seeing one todo item in our app? We’ve set the initial state value, but haven’t yet used the todo state value in our return statement. To show the todos, we need to loop through the todos array and render a todo item for each item inside of the todos array. For this, we’ll use the map function.
...

{todos.map((todo, i) => (
{todo.content}
))}

... Map is a very common function. You’ll likely come across it many times in your React career, so it’s important you understand how it works. To learn more about it, check out this guide. The code above is looping through the todos array and rendering the HTML inside of our parentheses for each item in the array.
Save the component, jump over to your browser and you’ll see something that looks like this: Create a New Todo Now that we’re able to display a list of todos in our React app, let’s add the ability to create a new todo item next. Other todo apps require you to click a button to create a new todo. I’m not a fan of that UX. It feels clunky. 57436996911.pdf We’re better UX designers than that! Instead, we’ll detect when the return key is pressed to create a new todo. 50699990246.pdf It feels more natural that way. Start by adding an onKeyDown event handler to the input field: ...
{todo.content} handleKeyDown(e, i)} />
... onKeyDown calls a function called handleKeyDown. It passes in the input’s event and the index of the todo.
Inside of handleKeyDown, we detect if the return key is pressed. If it is, we call createTodoAtIndex. Let’s add both of these functions above the return statement, like so: ... function handleKeyDown(e, i) { if (e.key === 'Enter') { createTodoAtIndex(e, i); } } function createTodoAtIndex(e, i) { const newTodos = [...todos]; newTodos.splice(i + 1, 0, { content: '', isCompleted: false, }); setTodos(newTodos); setTimeout(() => { document.forms[0].elements[i + 1].focus(); }, 0); } ... The createTodoAtIndex function looks complicated, but it’s actually quite simple. Let me break it down: We begin by detecting if the Return key is pressed by checking the value of event.key.Next, we create a copy of the todos state array. We do this because state should never be directly mutated (modified).Using the copy of todos, we insert a new empty todo after the currently selected todo. That’s why we needed to pass in the current todo index into this function.After inserting the new todo into our todos copy, we update the original todos array with the copy.Finally, we set the focus to the new input field. You may have noticed that I wrapped the line of code to focus on the new input field inside of a timeout that triggers after 0 milliseconds... ...huh?! Let me explain. Updating the state inside of a React component does not happen instantaneously. It can sometimes take time, especially if what we’re updating contains a lot of data. caregiver home health care daily log template Therefore, we add a timeout delay to the focus to wait for the state to finish updating before focusing on the newly rendered input. If you’d like to learn more about setTimeout, read my tutorial on setTimeout in React Components Using Hooks. Update a Todo Now that we can create a new todo, let’s add the functionality to actually write in a value for that todo item. Input fields have an onChange event handler which is triggered whenever the value of that field changes. Be careful though, as the value itself is not provided from the change handler. Instead, an event object is given, which allows you to find the value through event.target.value. Add the following function below the createTodoAtIndex function you created earlier: ... function updateTodoAtIndex(e, i) { const newTodos = [...todos]; newTodos[i].content = e.target.value; setTodos(newTodos); } ...
Much like createTodoAtIndex, updateTodoAtIndex takes two parameters: the input event and the todo index. Similarly, we make a copy of the todos array to avoid directly mutating the state. Within this copy, we change the value of the todo content key with the value inside of the event object. Finally, we update the todos state object with the updated copy, and, voila! Remove a Todo Remember our onKeyDown handler? We’re headed back there, but this time to detect if the backspace key is pressed. If it is, we call a new function called removeTodoAtIndex. function handleKeyDown(e, i) { if (e.key === 'Enter') { createTodoAtIndex(e, i); } if (e.key === 'Backspace' && todos[i].content === '') { e.preventDefault(); return removeTodoAtIndex(i); } } function removeTodoAtIndex(i) { if (i === 0 && todos.length === 1) return; setTodos(todos => todos.slice(0, i).concat(todos.slice(i + 1, todos.length))); setTimeout(() => { document.forms[0].elements[i - 1].focus(); }, 0); } Save the component, jump back over to your browser and delete one of the todo items. coleman saluspa 15442 manual You should see the whole todo item disappear when you hit backspace on an empty todo. Complete a Todo We’ve covered creating, updating, and deleting a todo. comparative and superlative adjectives worksheets printable Now, let’s add the last piece of the puzzle in our React todo app: completing a todo! Right now, when you hover over a circle inside of a todo, you’ll see it turn grey. That’s the CSS hover effect coming in to play. However, if you click the circle nothing happens. Let’s change that. Add an onClick handler to the todo as well as the todo-is-completed CSS class. ...
toggleTodoCompleteAtIndex(i)}> {todo.isCompleted && (✔)}
{todo.content} handleKeyDown(e, i)} onChange={e => updateTodoAtIndex(e, i)} />

… Finally, add the toggleTodoCompletedAtIndex function underneath the other functions: function toggleTodoCompleteAtIndex(index) { const temporaryTodos = [...todos]; temporaryTodos[index].isCompleted = !temporaryTodos[index].isCompleted; setTodos(temporaryTodos); } Save the component, open the React app, and click on one of the todo items’ checkboxes… The Full Source for React Todo I’ve provided the full source code below so you can see our React todo component in all its glory.

Don’t forget to follow me on Twitter for more original React tutorials like this one! import React, { useState } from 'react'; import logo from './logo.svg'; import './App.css'; function App() { const [todos, setTodos] = useState([{ content: 'Pickup dry cleaning', isCompleted: true, }, { content: 'Get haircut', isCompleted: false, }, { content: 'Build a todo app in React', isCompleted: false, }]); function handleKeyDown(e, i) { if (e.key === 'Enter') { createTodoAtIndex(e, i); } if (e.key === 'Backspace' && todos[i].content === '') { e.preventDefault(); return removeTodoAtIndex(i); } } function createTodoAtIndex(e, i) { const newTodos = [...todos]; newTodos.splice(i + 1, 0, { content: '', isCompleted: false, }); setTodos(newTodos); setTimeout(() => { document.forms[0].elements[i + 1].focus(); }, 0); } function updateTodoAtIndex(e, i) { const newTodos = [...todos]; newTodos[i].content = e.target.value; setTodos(newTodos); } function removeTodoAtIndex(i) { if (i === 0 && todos.length === 1) return; setTodos(todos => todos.slice(0, i).concat(todos.slice(i + 1, todos.length))); setTimeout(() => { document.forms[0].elements[i - 1].focus(); }, 0); } function toggleTodoCompleteAtIndex(index) { const temporaryTodos = [...todos]; temporaryTodos[index].isCompleted = !temporaryTodos[index].isCompleted; setTodos(temporaryTodos); } return (

```
    {todos.map((todo, i) => (
    toggleTodoCompleteAtIndex(i)}> {todo.isCompleted && ( ✔ )}
    {todo.content} handleKeyDown(e, i)} onChange={e => updateTodoAtIndex(e, i)} />
    ))}
```

); } export default App;