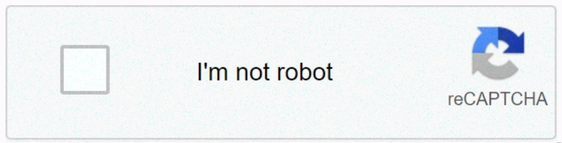


I'm not robot!

Popper.js sample code

Popper.js apply style example. What is popper.js used for. Popper.js dropdown examples. Popper js example.

In this tutorial you'll learn how to use Popper by building a basic tooltip.Remember, Popper is not a tooltip library, it's the foundation to build one! Follow the tutorial below to learn how to use Popper.Don't mind tech-related ads? Consider disabling your ad-blocker to help us!They are small and unobtrusive.Alternatively, support us on Open Collective!Setting upCreate a new HTML document with two elements, a `and a tooltip
, and pass them to Popper:` `My button` StylingLet's give our tooltip some styling: Here's the result so far:ArrowOur tooltip is currently just a box.

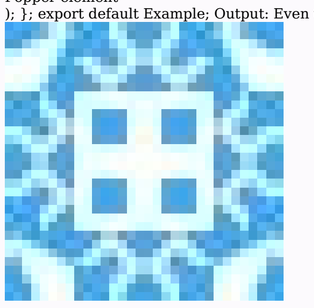
In many UI design systems, this is all tooltips need, but others prefer to have an arrow that points toward the reference element.Add an arrow element with a data-popper-arrow attribute like so:Now for styling:#arrow, #arrow::before { position: absolute; width: 8px; height: 8px; background: inherit; } #arrow { visibility: hidden; } #arrow::before { visibility: visible; content: ''; transform: rotate(45deg); }The ::before pseudo-element is required because Popper uses transform to position the arrow inside the popper, but we want to use our own transform to rotate the arrow box into a diamond.Now we need to offset the arrow depending on the popper's current placement. Popper provides this information with the data-popper-placement attribute:#tooltip[data-popper-placement^='top'] > #arrow { bottom: -4px; } #tooltip[data-popper-placement^='bottom'] > #arrow { top: -4px; } #tooltip[data-popper-placement^='left'] > #arrow { right: -4px; } #tooltip[data-popper-placement^='right'] > #arrow { left: -4px; }Why the ^=? This means "starts with", because we can also have variation placements like top-start.Here's the result so far:OffsetOur arrow currently overlaps the reference, we can give it a distance of 8px using the offset modifier:const popperInstance = Popper.createPopper(button, tooltip, { modifiers: [{ name: 'offset', options: { offset: [0, 8], }, },], });Here's the result so far:FunctionalityWe only want our tooltip to show when hovering or focusing the button.#tooltip { display: none; } #tooltip[data-show] { display: block; }function show() { tooltip.setAttribute('data-show', ''); popperInstance.update(); } function hide() { tooltip.removeAttribute('data-show'); } const showEvents = ['mouseenter', 'focus']; const hideEvents = ['mouseleave', 'blur']; showEvents.forEach((event) => { button.addEventListener(event, show); }); hideEvents.forEach((event) => { button.addEventListener(event, hide); });Here's the final result:PerformanceOnce we create a popper with createPopper, it stays "alive". This means while scrolling the page, the popper is constantly being updated, even if it is not visible. We can disable the event listeners when the tooltip is hidden to optimize it:function show() { tooltip.setAttribute('data-show', ''); popperInstance.setOptions((options) => ({ ...options, modifiers: [...options.modifiers, { name: 'eventListeners', enabled: true },], })); popperInstance.update(); } function hide() {

tooltip.removeAttribute('data-show'); popperInstance.setOptions((options) => ({ ...options, modifiers: [...options.modifiers, { name: 'eventListeners', enabled: false },], })); }Full code `My button` FinishedYou've now created a basic tooltip using Popper! Of course, there is more you can do, such as adding animations or interactivity. These are up to

you to explore when creating abstractions for common popper elements like tooltips, popovers, drop-downs, and more.Edit this page So my team and I are trying to create our own reusable UI component library that's not based on any UI frameworks and everything was butter until we came to the dropdown component. Dropdowns and modals are notoriously abstract because the elements in the DOM are not immediately nested. In order to have modals & dropdowns appear above all other elements (standard modal & dropdown behavior), you have to use reasonably advanced concepts. As I was looking for examples on the web, I ran into Popper.js. Great! A tooltip & popover positioning library. Just what we need. Most of the popper docs are written in pure vanilla JS. They have a very small section with limited details on using the react-popper. I plan to PR some doc additions to the lib. In their docs, they explain that hooks are the way forward (yay, we all love hooks... right?). So I start trying to implement the hooks example: Code Story borrowed straight from docs example Code: import React, { useState } from "react"; import { usePopper } from "react-popper"; const Example = () => { const [referenceElement, setReferenceElement] = useState(null); const [popperElement, setPopperElement] = useState(null); const [arrowElement, setArrowElement] = useState(null); const

{ styles, attributes } = usePopper(referenceElement, popperElement, { modifiers: [{ name: "arrow", options: { element: arrowElement } }] }); return (<> `Reference element`

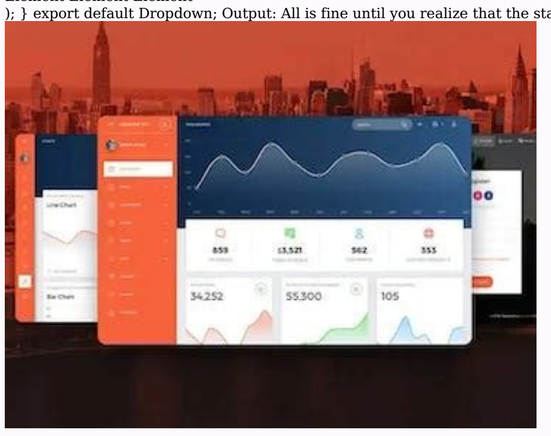
Popper element
); }; export default Example; Output: Even though styles are missing, I understand that the default docs example should be as vanilla as possible. This example doesn't visually do anything.



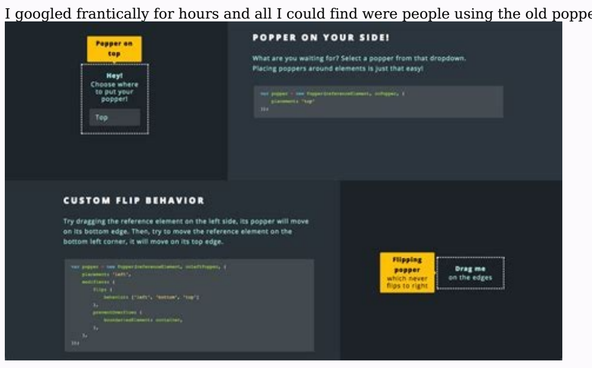
So I tried to implement this. Docs converted to dropdown Code: import React, { useState } from "react"; import { usePopper } from "react-popper"; import DropdownContainer from "./components/DropdownContainer"; import DropdownItem from "./components/DropdownItem"; function Dropdown(props) { const [visible, setVisibility] = useState(false); const [referenceRef, setReferenceRef] = useState(null); const [popperRef, setPopperRef] = useState(null); const { styles, attributes } = usePopper(referenceRef, popperRef, { placement: "bottom", modifiers: [{ name: "offset", enabled: true, options: { offset: [0, 10] } }] }); function handleDropdownClick(event) { setVisibility(!visible);

} return (`Click Me`

Element Element Element
); } export default Dropdown; Output: All is fine until you realize that the standard dropdown behavior is to close the dropdown on document click outside of your element. I could not find information in the popper docs ANYWHERE about this.



I googled frantically for hours and all I could find were people using the old popper style (Manager, Provider, render props, etc). I was determined to get the hooks example to work.



After all, hooks are the way forward. As it turns out, the generally accepted way to handle closing a dropdown or modal on click outside your component was a document event listener where you check to see if the click target includes your element. After wrangling with React's refs and implementing a document body click listener, here's where I landed: Final Result Code Code: import React, { useState, useEffect, useRef } from "react"; import { usePopper } from "react-popper"; import styled from "styled-components"; function Dropdown(props) { const [visible, setVisibility] = useState(false); const referenceRef = useRef(null); const popperRef = useRef(null); const { styles, attributes } = usePopper(referenceRef.current, popperRef.current, { placement: "bottom", modifiers: [{ name: "offset", enabled: true, options: { offset: [0, 10] } }] }); useEffect(() => { // listen for clicks and close dropdown on body document.addEventListener("mousedown", handleDocumentClick); return () => { document.removeEventListener("mousedown",

handleDocumentClick); }; }, []); function handleDocumentClick(event) { if (referenceRef.current.contains(event.target)) { return; } setVisibility(false); } function handleDropdownClick(event) { setVisibility(!visible); } return (`Click Me`

Element Element Element
); } const DropdownContainer = styled.div` display: ${props => (props.visible ?
"flex" : "none")}; width: "2px"; flex-direction: column; background-color: "#FFF"; border-radius: 4px; box-shadow: 0 0 8px 0 rgba(0, 0, 0, 0.14); padding: 5px; `; const DropdownItem = styled.div` justify-content: flex-start; height: 40px; padding-right: 10px; padding-left: 10px; align-items: center; &:hover { background-color: #00ffff; } &:active { font-weight: 700; color: #00ffff; } `; export default Dropdown; The important thing worth mentioning is that I used useRef instead of useState when creating refs which caused the actual ref objects to be accessed from referenceRef.current and popperRef.current. Hopefully, this saves you time, headaches, and by translation, money! Mykhailo Kokadii - Jun 16 Once suspended, tannerhallman will not be able to comment or publish posts until their suspension is removed. Once unsuspended, tannerhallman will be able to comment and publish posts again. Once unpublished, all posts by tannerhallman will become hidden and only accessible to themselves. If tannerhallman is not suspended, they can still re-publish their posts from their dashboard. Note: Thanks for keeping DEV Community safe. Here is what you can do to flag tannerhallman: Make all posts by tannerhallman less visible tannerhallman consistently posts content that violates DEV Community's code of conduct because it is harassing, offensive or spammy. Report other inappropriate conduct Unflagging tannerhallman will restore default visibility to their posts.