Frontend Technical Assessment

Description

You are to create an `Autocomplete` component, similar to those you may find in modern UI libraries but with just the bare minimum. An implemented example can be found here.

You will be scored based on requirements outlined below and code quality of the implemented component and its usage.

Requirements

Development Environment

The following libraries and frameworks must be used:

- React
- Typescript

You are also strongly encouraged to use the following:

- Tailwind CSS
- Floating UI (for positioning of the options window)

Other than the above, your component should not need any other dependencies.

Otherwise, you are free to use any build tool of your choice to work with and showcase your work. If you'd like to keep things quick and simple, you can just use Vite's `react-ts` template preset to get started with development.

Appearance and Styling

You may use the example component linked in the description as a baseline and are free to add your own touches or changes as needed.

(Bonus) Come up with a design to display the selected option(s) within the search input area.

Props

At minimum, the component would take in the following props:

Name	Type(s)	Description
`description`	`string`	The description to display below the component.
`disabled`	`boolean`	If `true`, the component is disabled, i.e. it cannot be interacted with.
`filterOptions`	`function`	A function to determine the filtered options to be rendered on search. If provided, this will override the default built-in filtering of the component. **Note**: If `options` of `string` type are provided, then the default filtering should just be a simple string equality comparison with the current input value. However, if `options` of `object` type are provided, you should determine an appropriate default built-in filtering for this scenario.
`label`	`string`	The label to display above the component.
`loading`	`boolean`	If `true`, the component will be in a loading state. This should show a spinner.
`multiple`	`boolean`	If `true`, then `value` must be an array and the multiple selections should be supported.
`onChange`	`function`	The callback that is fired when the value changes.
`onInputChange`	`function`	The callback that is fired when the input value changes.
`options`*	`Array<T>`	Array of options to be displayed and selected. **Note**: `T` can be either a `string` or an `object`.
`placeholder`	`string`	The placeholder search input text.
`renderOption`	`function`	Customises the rendered option display.
`value`	`Array<T>, T`	The selected value (if any) of the autocomplete

*Required

You are otherwise free to add any other props as needed. If you decide to modify or remove any of the above props, please indicate and explain these changes in a `README` file.

Controls

At minimum, the component should be able to be controlled via the following controls:

- Mouse
    - Clicking on the component opens up the options window and focuses on the search input.
    - Clicking on an option will (de)select the option.
    - When the options window is open, clicking outside the component will cause the window to close and unfocus the search input.
- Keyboard
    - Options can be iterated through via Up and Down Arrow Keys, and should be “loop around” at the start and end.
    - Option can be (de)selected via the Enter Key.
    - The options window can be closed via the Escape Key.

Component Usage

With your component, you should also be able to implement the following use-cases:

- Synchronous autocomplete with single and multiple option(s) selection.
- Debounced search, i.e. filtering of displayed options only occurs after typing has ceased for a specified amount of time.