



find packages

sign up or log in ★ 

# react-autosuggest

 publicbuild **passing** contributors **35** coverage **95%**downloads **199k/month** npm **v9.3.2** gzip size **7.78 kB**

## Demo


Check out the [Homepage](#) and the [Codepen examples](#).

## Features

- **WAI-ARIA compliant**, with support for ARIA attributes and keyboard interactions
- Mobile friendly
- Plugs in nicely to Flux and **Redux** applications
- Full control over **suggestions rendering**
- Suggestions can be presented as **plain list** or **multiple sections**
- Suggestions can be retrieved **asynchronously**
- **Highlight the first suggestion** in the list if you wish
- Supports styling using **CSS Modules**, **Radium**, **Aphrodite**, **JSS**, and more
- You decide **when to show suggestions** (e.g. when user types 2 or more characters)

## Unleash awesomeness

Private packages, team management tools, and powerful integrations. [Get started with npm Orgs](#)

 npm i react-autosuggest  
[how?](#) [learn more](#)



**moroshko** published a ...

**9.3.2** is the latest of 116 rel...

[github.com/moroshko/reac...](#)

MIT 

## Collaborators [list](#)



## Stats

**8 501** downloads in the last...

- **Always render suggestions** (useful for mobile and modals)
- **Pass through arbitrary props to the input** (e.g. placeholder, type, **onChange**, **onBlur**, or any other), or **take full control on the rendering of the input** (useful for integration with other libraries)
- Thoroughly tested

## Installation

```
yarn add react-autosuggest
```

or

```
npm install react-autosuggest --save
```

You can also use the standalone UMD build:

```
<script src="https://unpkg.com/react-autosu
```



## Basic Usage

```
import Autosuggest from 'react-autosuggest'
```

```
// Imagine you have a list of languages tha
const languages = [
```

```
{
  name: 'C',
  1070
```

41 999 downloads in the la...

199 093 downloads in the l...

52 open issues on GitHub

7 open pull requests on Git...

Try it out

Test react-autosuggest ...

## Keywords

autosuggest, autocomplete, auto-suggest, auto-complete, auto suggest, auto complete, react autosuggest, react autocomplete, react auto-suggest, react auto-complete, react auto suggest, react auto complete, react-autosuggest, react-autocomplete, react-auto-suggest, react-auto-complete, react-component

## Dependencies (3)

prop-types, react-autowhatever, shallow-equal

## Dependents (103)

@navrin/react-chips, react-components-form, cboard,

```

      year: 1972
    },
    {
      name: 'Elm',
      year: 2012
    },
    ...
  ];

  // Teach Autosuggest how to calculate suggestions
  const getSuggestions = value => {
    const inputValue = value.trim().toLowerCase
    const inputLength = inputValue.length;

    return inputLength === 0 ? [] : languages
      .filter(lang => lang.name.toLowerCase().slice(0, inputLength) === value)
      .slice(0, 5);
  };

  // When suggestion is clicked, Autosuggest
  // will update the input value based on the clicked suggestion. Teach Autosuggest
  // how to calculate the input value for every given suggestion.
  const getSuggestionValue = suggestion => suggestion.name;

  // Use your imagination to render suggestions for the Autosuggest component.
  const renderSuggestion = suggestion => (
    <div>
      {suggestion.name}
    </div>
  );

  class Example extends React.Component {
    constructor() {

```

seek-style-guide, zenith-common, ba-react-fetch-list-api, azsearch.js, catalyst-ui, react-github-field, jane-maps, rm3-tag-control, laji-form, web-modules-core, ims-shared-client, team-directory, smoke-editor, react-chips, rebass-autosuggest, react-django-selectable, react-frontend, zooid-ui-device-picker, samplereactcomp, react-bootstrap-table-mithun, candidate2, coapps-framework, @gogeo/react-autosuggest, nlx-react-common, mdlReact, true-mnp, redis-live, drw-react, typography-design-tools, nukleus, netlify-cms, wix-style-react, bpk-component-autosuggest, react-autosuggest-geocoder, uol-frontend-framework, oddsmarket-site, labo-components, react-generic, @kupibilet/ui, coderbox-components, @appbaseio/reactivebase, zenith-workers, uqlibrary-react-toolbox, @coderbox/components, ucsc-

```

super();

// Autosuggest is a controlled componen
// This means that you need to provide
// and an onChange handler that updates
// Suggestions also need to be provided
// and they are initially empty because
this.state = {
  value: '',
  suggestions: []
};
}

onChange = (event, { newValue }) => {
  this.setState({
    value: newValue
  });
};

// Autosuggest will call this function ev
// You already implemented this logic abc
onSuggestionsFetchRequested = ({ value })
  this.setState({
    suggestions: getSuggestions(value)
  });
};

// Autosuggest will call this function ev
onSuggestionsClearRequested = () => {
  this.setState({
    suggestions: []
  });
};

```

xena-client, style-panel, flex-editor, and more

[Guevara](#) is hiring. View more...

```

render() {
  const { value, suggestions } = this.state

  // Autosuggest will pass through all the props
  const inputProps = {
    placeholder: 'Type a programming language',
    value,
    onChange: this.onChange
  };

  // Finally, render it!
  return (
    <Autosuggest
      suggestions={suggestions}
      onSuggestionsFetchRequested={this.onSuggestionsFetchRequested}
      onSuggestionsClearRequested={this.onSuggestionsClearRequested}
      getSuggestionValue={this.getSuggestionValue}
      renderSuggestion={this.renderSuggestion}
      inputProps={inputProps}
    />
  );
}

```

## Props

prop	type	required	description

suggestions	Array	✓	These are the suggestions that will be displayed. Items can take an arbitrary shape.
onSuggestionsFetchRequested	Function	✓	Will be called every time you need to recalculate suggestions.
onSuggestionsClearRequested	Function	✓*	Will be called every time you need to set suggestions to [].
getSuggestionValue	Function	✓	Implement it to teach Autosuggest what should be the input value when suggestion is clicked.
renderSuggestion	Function	✓	Use your imagination to define how suggestions are rendered.

<code>inputProps</code>	Object	✓	Pass through arbitrary props to the input. It must contain at least <code>value</code> and <code>onChange</code> .
<code>onSuggestionSelected</code>	Function		Will be called every time suggestion is selected via mouse or keyboard.
<code>onSuggestionHighlighted</code>	Function		Will be called every time the highlighted suggestion changes.
<code>shouldRenderSuggestions</code>	Function		When the input is focused, Autosuggest will consult this function when to render suggestions. Use it, for example, if you want to display suggestions when input value is at least 2 characters long.

<code>alwaysRenderSuggestions</code>	Boolean		Set it to <code>true</code> if you'd like to render suggestions even when the input is not focused.
<code>highlightFirstSuggestion</code>	Boolean		Set it to <code>true</code> if you'd like Autosuggest to automatically highlight the first suggestion.
<code>focusInputOnSuggestionClick</code>	Boolean		Set it to <code>false</code> if you don't want Autosuggest to keep the input focused when suggestions are clicked/tapped.
<code>multiSection</code>	Boolean		Set it to <code>true</code> if you'd like to display suggestions in multiple sections (with optional titles).



<code>renderSectionTitle</code>	Function	✓ when <code>multiSection={true}</code>	Use your imagination to define how section titles are rendered.
<code>getSectionSuggestions</code>	Function	✓ when <code>multiSection={true}</code>	Implement it to teach Autosuggest where to find the suggestions for every section.
<code>renderInputComponent</code>	Function		Use it only if you need to customize the rendering of the input.
<code>renderSuggestionsContainer</code>	Function		Use it if you want to customize things inside the suggestions container beyond rendering the suggestions themselves.
<code>theme</code>	Object		Use your imagination to style the Autosuggest.

<code>id</code>	String	Use it only if you have multiple Autosuggest components on a page.
-----------------	--------	--

### **suggestions (required)**

Array of suggestions to display. The only requirement is that `suggestions` is an array. Items in this array can take an arbitrary shape.

For a plain list of suggestions, every item in `suggestions` represents a single suggestion. It's up to you what shape every suggestion takes. For example:

```
const suggestions = [  
  {  
    text: 'Apple'  
  },  
  {  
    text: 'Banana'  
  },  
  {  
    text: 'Cherry'  
  },  
  {  
    text: 'Grapefruit'  
  },  
  {  
    text: 'Lemon'  
  }  
];
```

For **multiple sections**, every item in suggestions represents a single section. Again, it's up to you what shape every section takes. For example:

```
const suggestions = [  
  {  
    title: 'A',  
    suggestions: [  
      {  
        id: '100',  
        text: 'Apple'  
      },  
      {  
        id: '101',
```

```

        text: 'Apricot'
    }
]
},
{
    title: 'B',
    suggestions: [
        {
            id: '102',
            text: 'Banana'
        }
    ]
},
{
    title: 'C',
    suggestions: [
        {
            id: '103',
            text: 'Cherry'
        }
    ]
}
];

```

### onSuggestionsFetchRequested (required)

This function will be called every time you might need to update **suggestions**. It has the following signature:

```
function onSuggestionsFetchRequested({ valu
```



where:

- `value` - the current value of the input
  - `reason` - string describing why `onSuggestionsFetchRequested` was called. The possible values are:
    - `'input-changed'` - user typed something
    - `'input-focused'` - input was focused
    - `'escape-pressed'` - user pressed Escape to clear the input (and suggestions are shown for empty input)
    - `'suggestions-revealed'` - user pressed Up or Down to reveal suggestions
    - `'suggestion-selected'` - user selected a suggestion
- when `alwaysRenderSuggestions={true}`

### **`onSuggestionsClearRequested` (required unless `alwaysRenderSuggestions={true}`)**

This function will be called every time you need to clear **suggestions**.

All you have to do in this function is to set `suggestions` to `[]`.

**Note:** When `alwaysRenderSuggestions={true}`, you don't have to implement this function.

### **`getSuggestionValue` (required)**

When user navigates the suggestions using the Up and Down keys, **the input value should be set according to the highlighted suggestion**. You design how suggestion is modelled. Therefore, it's your responsibility to tell Autosuggest how to map suggestions to input values.

This function gets the suggestion in question, and it should return a string. For example:

```
function getSuggestionValue(suggestion) {  
  return suggestion.text;  
}
```

## renderSuggestion (required)

Use your imagination to define how suggestions are rendered.

The signature is:

```
function renderSuggestion(suggestion, { query
```



where:

- `suggestion` - The suggestion to render
- `query` - Used to highlight the matching string. As user types in the input, `query` will be equal to the trimmed value of the input. Then, if user interacts using the Up or Down keys, **the input will get the value of the highlighted suggestion**, but `query` will remain to be equal to the trimmed value of the input prior to the Up and Down interactions.
- `isHighlighted` - Whether or not the suggestion is highlighted.

It should return a string or a `ReactElement`. For example:

```
function renderSuggestion(suggestion) {  
  return (  
    <span>{suggestion.text}</span>  
  );  
}
```

**Important:** renderSuggestion must be a pure function (we optimize rendering performance based on this assumption).

### inputProps (required)

Autosuggest is a **controlled component**. Therefore, you MUST pass at least a value and an onChange callback to the input.

You can pass any other props as well. For example:

```
const inputProps = {  
  value,           // usually comes from the  
  onChange,        // called every time the  
  onBlur,          // called when the input  
  type: 'search',  
  placeholder: 'Enter city or postcode'  
};
```



### inputProps.onChange (required)

The signature is:

```
function onChange(event, { newValue, method
```



where:

- newValue - the new value of the input
- method - string describing how the change has occurred. The possible values are:
  - 'down' - user pressed Down
  - 'up' - user pressed Up
  - 'escape' - user pressed Escape
  - 'enter' - user pressed Enter

- 'click' - user clicked (or tapped) on suggestion
- 'type' - none of the methods above (usually means that user typed something, but can also be that they pressed Backspace, pasted something into the input, etc.)

### inputProps.onBlur (optional)

The signature is:

```
function onBlur(event, { highlightedSuggest
```



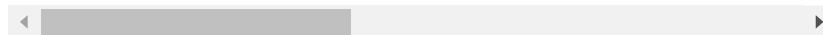
where:

- highlightedSuggestion - the suggestion that was highlighted just before the input lost focus, or null if there was no highlighted suggestion.

### onSuggestionSelected (optional)

This function is called when suggestion is selected. It has the following signature:

```
function onSuggestionSelected(event, { sugg
```



where:

- suggestion - the selected suggestion
- suggestionValue - the value of the selected suggestion (equivalent to getSuggestionValue(suggestion))
- suggestionIndex - the index of the selected suggestion in the suggestions array
- sectionIndex - when rendering **multiple sections**, this will be the section index (in **suggestions**) of the selected



suggestion. Otherwise, it will be null.

- `method` - string describing how user selected the suggestion.

The possible values are:

- `'click'` - user clicked (or tapped) on the suggestion
- `'enter'` - user selected the suggestion using Enter

### **onSuggestionHighlighted (optional)**

This function is called when the highlighted suggestion changes. It has the following signature:

```
function onSuggestionHighlighted({ suggestion
```



where:

- `suggestion` - the highlighted suggestion, or null if there is no highlighted suggestion.

### **shouldRenderSuggestions (optional)**

By default, suggestions are rendered when the input isn't blank. Feel free to override this behaviour.

This function gets the current value of the input, and it should return a boolean.

For example, to display suggestions only when input value is at least 3 characters long, do:

```
function shouldRenderSuggestions(value) {  
  return value.trim().length > 2;  
}
```

When `shouldRenderSuggestions` returns `true`, **suggestions will be rendered only when the input is focused**.

If you would like to render suggestions regardless of whether the input is focused or not, set `alwaysRenderSuggestions={true}` (`shouldRenderSuggestions` is ignored in this case).

### **`alwaysRenderSuggestions` (optional)**

Set `alwaysRenderSuggestions={true}` if you'd like to always render the suggestions.

**Important:** Make sure that the initial value of `suggestions` corresponds to the initial value of `inputProps.value`. For example, if you'd like to show all the suggestions when the input is empty, your initial state should be something like:

```
this.state = {  
  value: '',  
  suggestions: allSuggestions  
};
```

### **`highlightFirstSuggestion` (optional)**

When `highlightFirstSuggestion={true}`, `Autosuggest` will automatically highlight the first suggestion. Defaults to `false`.

### **`focusInputOnSuggestionClick` (optional)**

By default, `focusInputOnSuggestionClick={true}`, which means that, every time suggestion is clicked (or tapped), the input keeps the focus.

On mobile devices, when the input is focused, the native keyboard appears. You'll probably want to lose the focus when

suggestion is tapped in order to hide the keyboard.

You can do something like this:

```
<Autosuggest focusInputOnSuggestionClick={ !
```



where `isMobile` is a boolean describing whether `Autosuggest` operates on a mobile device or not. You can use [kaimallea/isMobile](#), for example, to determine that.

### **multiSection (optional)**

By default, `Autosuggest` renders a plain list of suggestions.

If you'd like to have multiple sections (with optional titles), set `multiSection={true}`.

### **renderSectionTitle (required when `multiSection={true}`)**

When rendering **multiple sections**, you need to tell `Autosuggest` how to render a section title.

This function gets the section to render (an item in the **suggestions** array), and it should return a string or a `ReactElement`. For example:

```
function renderSectionTitle(section) {  
  return (  
    <strong>{section.title}</strong>  
  );  
}
```

If `renderSectionTitle` returns `null` or `undefined`, `section` title is not rendered.

### **`getSectionSuggestions` (required when `multiSection={true}`)**

When rendering **multiple sections**, you need to tell Autosuggest where to find the suggestions for a given section.

This function gets the section to render (an item in the **suggestions** array), and it should return an array of suggestions to render in the given section. For example:

```
function getSectionSuggestions(section) {  
  return section.suggestions;  
}
```

**Note:** Sections with no suggestions are not rendered.

### **`renderInputComponent` (optional)**

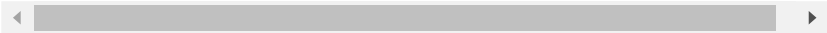
You shouldn't specify `renderInputComponent` unless you want to customize the rendering of the input.

To keep Autosuggest **accessible**, `renderInputComponent` **MUST**:

- render an input
- pass through all the provided `inputProps` to the input

Example:

```
const renderInputComponent = inputProps =>
  <div>
    <input {...inputProps} />
    <div>custom stuff</div>
  </div>
);
```



**Note:** When using `renderInputComponent`, you still need to specify the usual **inputProps**. Autosuggest will merge the `inputProps` that you provide with other props that are needed for accessibility (e.g. `'aria-activedescendant'`), and will pass the **merged inputProps** to `renderInputComponent`.

### **renderSuggestionsContainer (optional)**

You shouldn't specify `renderSuggestionsContainer` unless you want to customize the content or behaviour of the suggestions container beyond rendering the suggestions themselves. For example, you might want to add a custom text before/after the suggestions list, or to **customize the scrolling behaviour of the suggestions container**.

The signature is:

```
function renderSuggestionsContainer({ conta
```



where:

- `containerProps` - props that you MUST pass to the topmost element that is returned from `renderSuggestionsContainer`.

- children - the suggestions themselves. It's up to you where to render them.
- query - Same as query in **renderSuggestion**.

For example:

```
function renderSuggestionsContainer({ containerProps, children }) {  
  return (  
    <div {... containerProps}>  
      {children}  
      <div>  
        Press Enter to search <strong>{query}</strong>  
      </div>  
    </div>  
  );  
}
```



When `renderSuggestionsContainer` returns a composite component (e.g. `<IsolatedScroll ... />` as opposed to a DOM node like `<div ... />`), you **MUST** call `containerProps.ref` with the topmost element that the composite component renders.

For example:

```
import IsolatedScroll from 'react-isolated-
```

```
function renderSuggestionsContainer({ conta
  const { ref, ...restContainerProps } = co
  const callRef = isolatedScroll => {
    if (isolatedScroll !== null) {
      ref(isolatedScroll.component);
    }
  };

  return (
    <IsolatedScroll ref={callRef} {...restC
      {children}
    </IsolatedScroll>
  );
}
```



### theme (optional)

Autosuggest comes with no styles.

It uses **react-themeable** that allows you to style your Autosuggest component using **CSS Modules**, **Radium**, **Aphrodite**, **JSS**, **Inline styles**, and global CSS.

For example, to style the Autosuggest using CSS Modules, do:

```
/* theme.css */
```

```
.container { ... }  
.input { ... }  
.suggestionsContainer { ... }  
.suggestion { ... }  
.suggestionHighlighted { ... }  
...
```

```
import theme from 'theme.css';
```

```
<Autosuggest theme={theme} ... />
```

When not specified, theme defaults to:



```

{
  container: 'react-autosugg
  containerOpen: 'react-autosugg
  input: 'react-autosugg
  inputOpen: 'react-autosugg
  inputFocused: 'react-autosugg
  suggestionsContainer: 'react-autosugg
  suggestionsContainerOpen: 'react-autosugg
  suggestionsList: 'react-autosugg
  suggestion: 'react-autosugg
  suggestionFirst: 'react-autosugg
  suggestionHighlighted: 'react-autosugg
  sectionContainer: 'react-autosugg
  sectionContainerFirst: 'react-autosugg
  sectionTitle: 'react-autosugg
}

```



The following picture illustrates how theme keys correspond to Autosuggest DOM structure:

