TELUS Component Library

Autocomplete

Multi-Platform Component | Figma UI KIT

```
function AutocompleteExample() {
  const addresses = [
    '510 West Georgia Street, Vancouver, BC',
    '510 West Hastings Street, Vancouver, BC',
    '510 West Broadway, Vancouver, BC',
    '510 West 8th Avenue, Vancouver, BC',
    '510 West Queens Road, North Vancouver, BC'
  ]
  const initialItems = addresses.map((address) => ({ id:
address, label: address }))

  return (
    <div style={{ height: '300px' }}>
      <Autocomplete initialItems={initialItems} />
    </div>
  )
}
```

DANGER

The example provided above demonstrates an uncontrolled usage of autocomplete, which is designed specifically for web applications. If you are developing a React Native app, we recommend using the controlled usage of autocomplete.

Introduction

Use `Autocomplete` to provide suggestions based on user input, help reduce errors associated with long queries and typos and can minimize the chance of an unsuccessful search.

Follow the appropriate instructions to add this component in to your app.

Guidance

- Use a `TextInput` form field (default) or `Search` component (via a render function), so users can type freely
- Don't use `Autocomplete` as an alternative to a Select form field: `Autocomplete` provides hints and guidance aiding user input, while `Select` form field provides predetermined choices for user to select
- Keep the `Autocomplete` list manageable (you can use `maxSuggestions` prop to control its length)
- If the suggestions require more than 2 seconds to load, consider using the loading state of the component to provide feedback for the user that the system is processing their search
- Allow for forgiving text entry or fuzzy matching
- Show results after the user types at least 1 character (controlled via `minToSuggestion` prop)

Displaying supporting elements

Note that the `Autocomplete` component accepts the props to display supporting elements (like labels, hints, etc.) around the input, e.g.

Your address Please pick your address ?

Please make sure your address is correct

```
function AutocompleteSupportsExample() {
  const addresses = [
    '510 West Georgia Street, Vancouver, BC',
    '510 West Hastings Street, Vancouver, BC'
  ]
  const initialItems = addresses.map((address) => ({ id:
address, label: address }))

  return (
    <div style={{ height: '150px' }}>
      <Autocomplete
        initialItems={initialItems}
        label="Your address"
        hint="Please pick your address"
        tooltip="Use the list of suggestions to select your
address"
        feedback="Please make sure your address is correct"
      />
    </div>
  )
}
```

Using with a custom input

Even though `Autocomplete` component displays a `TextInput` by default, you can use it with a custom input component, for example:

Pick a country

```
function AutocompleteSupportsExample() {
  const countries = ['Canada', 'Mexico', 'United States']
  const initialItems = countries.map((country) => ({ id:
country, label: country }))

  return (
    <div style={{ height: '150px' }}>
      <Autocomplete
        initialItems={initialItems}
        children={({ inputRef, onChange, onKeyPress,
initialValue, value }) => (
          <Search
            onChange={onChange}
            onKeyPress={onKeyPress}
            ref={inputRef}
            initialValue={initialValue}
            placeholder="Pick a country"
            value={value}
          />
        )}
      />
    </div>
  )
}
```

Controlled usage

By default, if no `value` is provided, `Autocomplete` functions in an uncontrolled mode. This effectively means that the component itself analyzes the input value and filters the `initialItems` to provide suggestions. Once the user selects a suggestion from the list, the component will also call `onSelect` callback with the `id` of the selected suggestion.

Now, if you would like to to have more control over the workflow, you can use `Autocomplete` in a controlled mode. In order to do that, you need to keep the `items` and the current `value` of the input field in an outer state and adjust those in the `onChange` callback before passing to the component:

```
function AutocompleteControlledExample() {
  const addresses = [
    '510 West Georgia Street, Vancouver, BC',
    '510 West Hastings Street, Vancouver, BC',
    '510 West Broadway, Vancouver, BC',
    '510 West 8th Avenue, Vancouver, BC',
    '510 West Queens Road, North Vancouver, BC'
  ]
  const initialItems = addresses.map((address) => ({ id:
address, label: address }))
  const [value, setValue] = useState('')
  const [items, setItems] = useState(initialItems)
  const handleChange = (newValue) => {
    setValue(newValue)
    setItems(
      initialItems.filter(({ label }) =>
label.toLowerCase().includes(newValue.toLowerCase()))
    )
  }

  return (
    <div style={{ height: '300px' }}>
      <Autocomplete items={items} value={value} onChange=
{handleChange} />
    </div>
  )
}
```

This workflow is especially useful when one has to fetch suggestions from an API of some sort.

Nested usage

If you have nested items in your workflow, you can use `Autocomplete` in a controlled mode with each item having a `nested` property. In order to do that, you need to keep the `items` and the current `value` of the input field in an outer state and adjust those in the `onChange` and `onSelect` callback before passing to the component:

```
function AutocompleteNestedExample() {
  const addresses = [
    { label: '123 Main Street, Toronto, ON M5V 2J4', nested:
true, id: 1 },
    { label: '124 Kings Street, Toronto, ON M5V 2J4', nested:
false, id: 2 },
    { label: '35 Fraser Street, Toronto, ON M5V 2T4', nested:
false, id: 3 }
  ]

  const apartments = [
    { label: '1 - 123 Main Street, Toronto, ON M5V 2J4', nested:
false, id: 1 },
    { label: '2 - 123 Main Street, Toronto, ON M5V 2J4', nested:
false, id: 2 }
  ]

  const [value, setValue] = useState('')
  const [items, setItems] = useState(addresses)
  const handleChange = (newValue) => {
    setValue(newValue)
    setItems(addresses.filter(({ label }) =>
label.toLowerCase().includes(newValue.toLowerCase())))
  }
  const handleSelect = (id) => {
    const item = items.find((item) => item.id === id)
    if (item.nested) {
      setItems(apartments)
    }
  }
  return (
    <Autocomplete
      initialItems={undefined}
      items={items}
      value={value}
      onChange={handleChange}
      onSelect={handleSelect}
    />
  )
}
```

Displaying a loading state

When the suggestions are being loaded from an external source, this process can take a while. In cases like that it can be useful to display a loading state indicator via the `isLoading` prop:

```
<div style={{ height: '128px' }}>
  <Autocomplete isLoading={true} onChange={() => {}} />
</div>
```

If no results are available

When there is no suggestions available, the component displays a message defined via the `noResults` prop:

```
<div style={{ height: '128px' }}>
  <Autocomplete noResults="No suggestions matching your query is
available" onChange={() => {}} />
</div>
```

Not that you can also pass some JSX in that prop, and it will be displayed instead of text:

```
<div style={{ height: '128px' }}>
  <Autocomplete
    noResults={<Link href="http://telus.com">No results found,
proceed to the home page</Link>}
    onChange={() => {}}
  />
</div>
```

Accessibility

`Autocomplete` has `combobox` role applied by default.

Props

Name	Type	Platform	Default	Description
copy	union	standard	'en'	Copy language identifier
fullWidth		standard	true	
isLoading	bool	standard	false	Set to true in order to display the loading indicator instead of results

Name	Type	Platform	Default	Description
maxSuggestions	number	standard	5	Maximum number of suggestions provided at the same time
minToSuggestion	number	standard	1	Minimum number of characters typed for a list of suggestions to appear
helpText	string	standard	''	Help text to display when the input is focused and empty
children	func	standard		Can be used to provide a function that renders a custom input: <Autocomplete items={items} value={currentValue}> {({ inputId, inputRef, onChange, onKeyPress, readOnly, tokens, value }) => (<Search nativeID={inputId} ref=

Name	Type	Platform	Default	Description
				{inputRef} onChange= {onChange} onKeyPress= {onKeyPress} readOnly= {readOnly} tokens={tokens} value={value} />)} </Autocomplete>
items	arrayOf	standard		List of items to display as suggestions
loadingLabel	string	standard		Label to display alongside the spinner when in a loading state
noResults	node	standard		Text or JSX to render when no results are available
onChange	func	standard		Handler function to be called when the input value changes
onClear	func	standard		Handler function to be called when the clear button (appears if the handler is

Name	Type	Platform	Default	Description
				passed) is pressed
onSelect	func	standard		Callback function to be called when an item is selected from the list
value	string	standard		Input value for controlled usage

Variants

This component does not have any stylistic variants.

Feedback

- Spotted a problem with this component? Raise an issue on GitHub
- See any existing issues for this component
- Contact the team on slack in #ds-support