Introduction

`user-event` is a companion library for Testing Library that simulates user interactions by dispatching the events that would happen if the interaction took place in a browser.

> **LATEST VERSION**
>
> These docs describe `user-event@14`. We recommend updating your projects to this version, as it includes important bug fixes and new features. You can find the docs for `user-event@13.5.0` here, and the changelog for the release here.

While most examples with `user-event` are for `React`, the library can be used with any framework as long as there is a DOM.

Differences from `fireEvent`

`fireEvent` dispatches *DOM events*, whereas `user-event` simulates full *interactions*, which may fire multiple events and do additional checks along the way.

Testing Library's built-in `fireEvent` is a lightweight wrapper around the browser's low-level `dispatchEvent` API, which allows developers to trigger any event on any element. The problem is that the browser usually does more than just trigger one event for one interaction. For example, when a user types into a text box, the element has to be focused, and then keyboard and input events are fired and the selection and value on the element are manipulated as they type.

`user-event` allows you to describe a user interaction instead of a concrete event. It adds visibility and interactability checks along the way and manipulates the DOM just like a user interaction in the browser would. It factors in that the browser e.g. wouldn't let a user click a

hidden element or type in a disabled text box.
This is why you should use `user-event` to test interaction with your components.

There are, however, some user interactions or aspects of these that aren't yet implemented and thus can't yet be described with `user-event`. In these cases you can use `fireEvent` to dispatch the concrete events that your software relies on.

Note that this makes your component and/or test reliant upon your assumptions about the concrete aspects of the interaction being correct. Therefore if you already put in the work to specify the correct aspects of such interaction, please consider contributing to this project so that `user-event` might cover these cases too.

Writing tests with `userEvent`

We recommend invoking `userEvent.setup()` before the component is rendered. This can be done in the test itself, or by using a setup function. We discourage rendering or using any `userEvent` functions outside of the test itself - e.g. in a `before`/`after` hook - for reasons described in "Avoid Nesting When You're Testing".

```
// inlining
test('trigger some awesome feature when clicking the button', async
() => {
  const user = userEvent.setup()
  render(<MyComponent />)

  await user.click(screen.getByRole('button', {name: /click
me!/i}))

  // ...assertions...
})
```

```
// setup function
function setup(jsx) {
  return {
```

```
    user: userEvent.setup(),
    ...render(jsx),
  }
}

test('render with a setup function', async () => {
  const {user} = setup(<MyComponent />)
  // ...
})
```

Note that, while directly invoking APIs such as `userEvent.click()` (which will trigger `setup` internally) is still supported in v14, this option exists to ease the migration from v13 to v14, and for simple tests. We recommend using the methods on the instances returned by `userEvent.setup()`.

Edit this page

*Last updated on **Jul 10, 2022** by **Andrew***