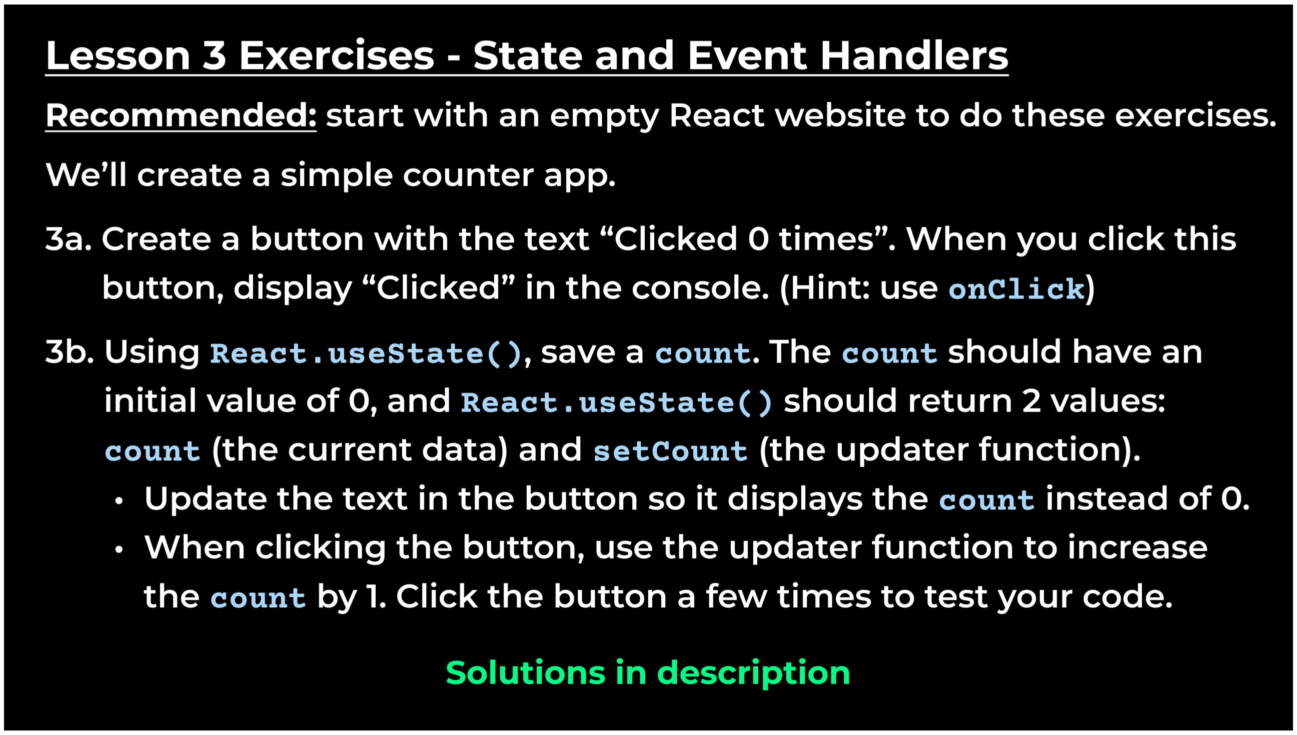


Lesson 3 Exercises - State and Event Handlers

Recommended: start with an empty React website to do these exercises.

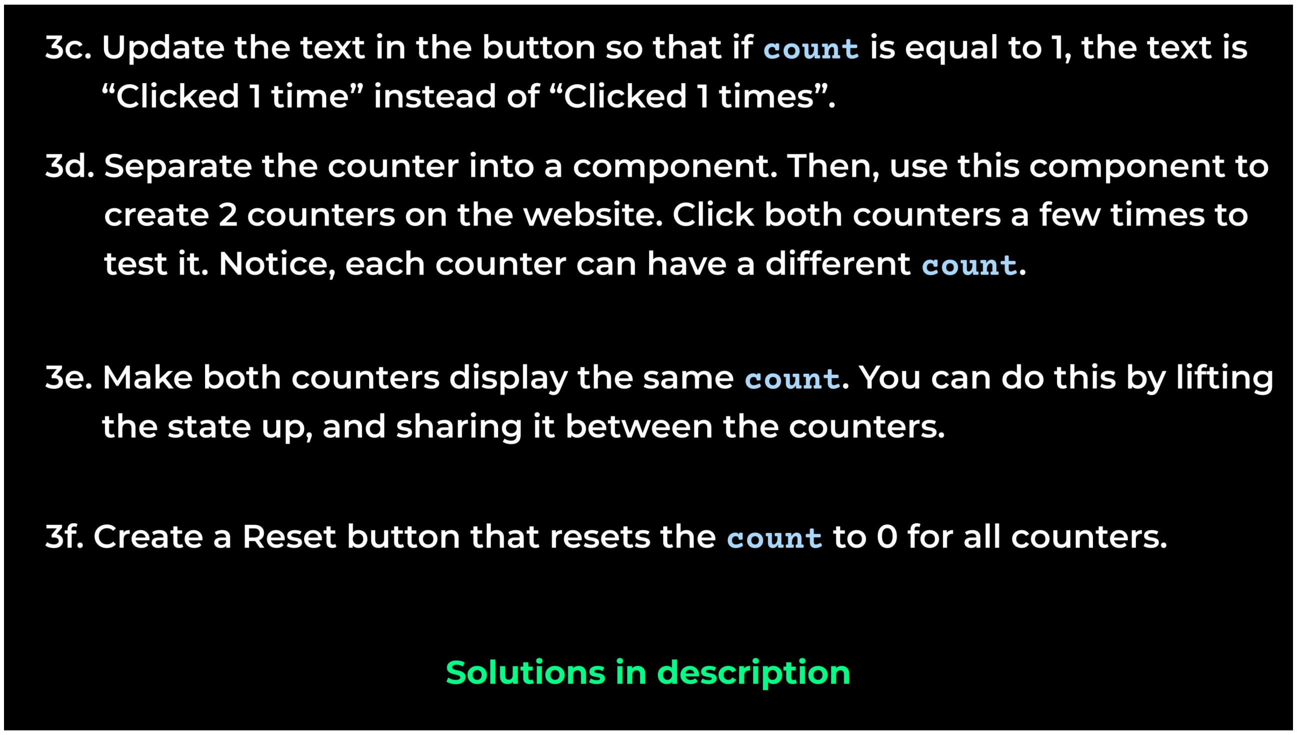
We'll create a simple counter app.

3a. Create a button with the text "Clicked 0 times". When you click this button, display "Clicked" in the console. (Hint: use `onClick`)

3b. Using `React.useState()`, save a `count`. The `count` should have an initial value of 0, and `React.useState()` should return 2 values: `count` (the current data) and `setCount` (the updater function).

- Update the text in the button so it displays the `count` instead of 0.
- When clicking the button, use the updater function to increase the `count` by 1. Click the button a few times to test your code.

Solutions in description



3c. Update the text in the button so that if `count` is equal to 1, the text is "Clicked 1 time" instead of "Clicked 1 times".

3d. Separate the counter into a component. Then, use this component to create 2 counters on the website. Click both counters a few times to test it. Notice, each counter can have a different `count`.

3e. Make both counters display the same `count`. You can do this by lifting the state up, and sharing it between the counters.

3f. Create a Reset button that resets the `count` to 0 for all counters.

Solutions in description

We'll create a display-as-you-type app.

3g. Start from an empty React website and create an `<input>` and a `<p>`. As you type in the `<input>`, the text you type should also appear in the `<p>` with "Hello " in front of it. (Hint: use `onChange`, save the text in state, and insert it in the JSX).

3h. Add 2 buttons beside the `<input>`, "Reset" and "Example".

- When clicking the Reset button, it removes the text in the `<input>`
- When clicking the Example button, it sets an example text in the `<input>`, like "Alice" (or any name you want).
- Hint: use a controlled input.

Challenge Exercises

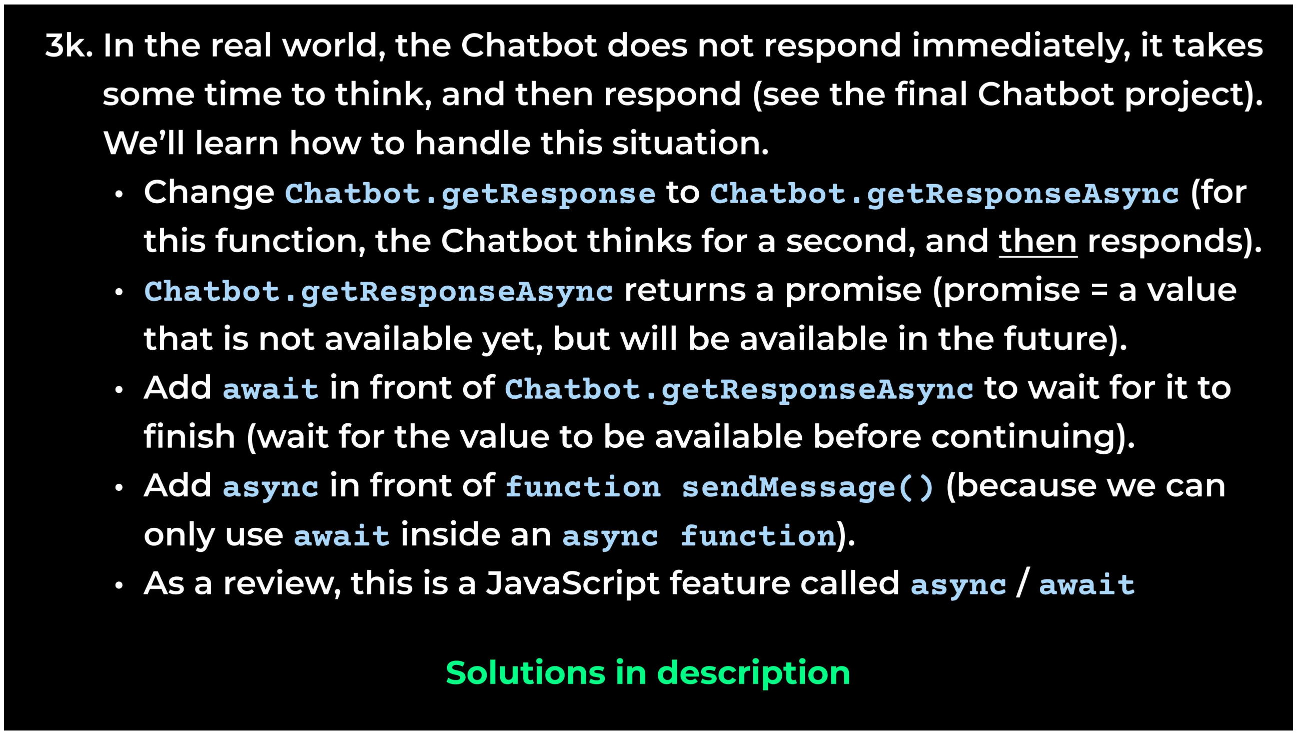
In the next exercises, we'll add code to `chatbot.html`

3i. Instead of clicking the "Send" button, we can also send a message by pressing "Enter" on our keyboard. We'll add this feature to the project.

- Add a new event prop `onKeyDown` to the `<input>`. This runs a function each time we press a key on our keyboard.
- The function gets an `event` parameter, and we can get the key that was pressed using `event.key`
- If `event.key` is `'Enter'`, send the message.

3j. If we type in the `<input>` and press "Esc" (or "Escape") it should remove the text in the `<input>`.

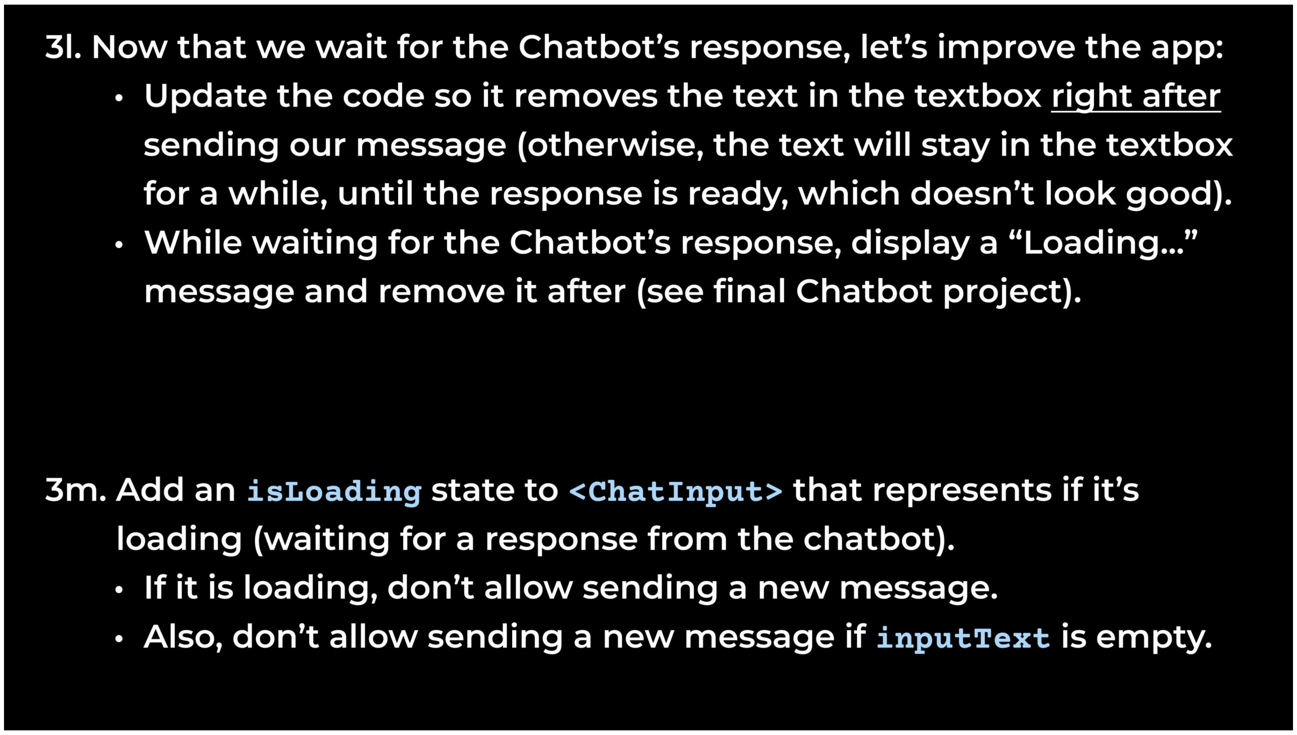
- Hint: check if `event.key` is `'Escape'`



3k. In the real world, the Chatbot does not respond immediately, it takes some time to think, and then respond (see the final Chatbot project). We'll learn how to handle this situation.

- Change `Chatbot.getResponse` to `Chatbot.getResponseAsync` (for this function, the Chatbot thinks for a second, and then responds).
- `Chatbot.getResponseAsync` returns a promise (promise = a value that is not available yet, but will be available in the future).
- Add `await` in front of `Chatbot.getResponseAsync` to wait for it to finish (wait for the value to be available before continuing).
- Add `async` in front of `function sendMessage()` (because we can only use `await` inside an `async function`).
- As a review, this is a JavaScript feature called `async` / `await`

Solutions in description



3l. Now that we wait for the Chatbot's response, let's improve the app:

- Update the code so it removes the text in the textbox right after sending our message (otherwise, the text will stay in the textbox for a while, until the response is ready, which doesn't look good).
- While waiting for the Chatbot's response, display a "Loading..." message and remove it after (see final Chatbot project).

3m. Add an `isLoading` state to `<ChatInput>` that represents if it's loading (waiting for a response from the chatbot).

- If it is loading, don't allow sending a new message.
- Also, don't allow sending a new message if `inputText` is empty.