DASHLANE

Forms handling w/ React: Giving Up Control

Tim

Senior Front-End Engineer @ Dashlane

twitter.com/tpillard

timtech.blog

Table of contents

Forms on the web

A document section containing interactive controls for submitting information.

HTML <form> element

- Markup
- Handling with JavaScript

HTML <form> element: markup

```
<form id="login-form">
  <label for="email">Email:</label>
  <input type="email" name="email" id="email" required>
  <label for="password">Password:</label>
  <input type="password" name="password" id="password" required>
  <button type="submit">Log-in</button>
</form>
```

HTML <form> element: handling with JavaScript

```
const form = document.querySelector('#login-form');
form.addEventListener('submit', function handleSubmit(evt) {
  evt.preventDefault();
  const formData = new FormData(evt.target);
  const values = {};
  for (let [name, value] of formData.entries()) {
    values[name] = value;
  }
  attemptUserLogIn(values);
});
```

Forms on the web with React

Forms ~~on the web~~ with React

~~Forms on the web with~~ React

“React allows you to express your UI as a function of its state.”

— Some person on the Internet

“When using React for the web, the DOM becomes an implementation detail.”

— Probably the same person on the Internet

Forms on the web with React

Controlled inputs

This is the way

Controlled inputs

This is the way

A controlled input is a form input which state is managed solely through React.

In other words, with a **controlled input**:

- The value is tracked, read, updated **by the app** (React State, Reducer, Redux, etc…)
- Every user input is handled through **React's Event System** to determine the new value.
- The default browser behavior is **suppressed**.
- The value is always set **as a prop**.
- Every value change triggers a **re-render** of the enclosing component.

Controlled input form example

A Form component with React using a controlled input

```
function Form() {
  const [value, setValue] = React.useState('');

  const handleChange = (evt) => {
    // Perform “on the fly” validation, transformation logic here if needed
    setValue(evt.target.value);
  };

  const handleSubmit = (evt) => {
    evt.preventDefault();
    performLogIn(value);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="password-input">Password:</label>
      <input id="password-input" type="password" value={value} onChange={handleChange} />
      <button type="submit">
        Submit
      </button>
    </form>
  );
}
```

Uncontrolled inputs

The old ways

Uncontrolled inputs

The old ways

A uncontrolled input is a form input which state is managed by the browser through the DOM.

In other words, with an **uncontrolled input**:

- In most cases, the app will only set the input's default value via the **defaultValue prop**.
- Every user input is **handled by the browser** to determine the new value, **according to input attributes**.
- The default browser behavior is **embraced**.
- **No superfluous React rendering.**
- **No extra state management required.**

Uncontrolled input form example

A Form component with React using an uncontrolled input

```
function Form() {
  const inputRef = React.useRef(null);

  const handleSubmit = (evt) => {
    evt.preventDefault();
    // Alternatively, use FormData or equivalent.
    const { current: input } = inputRef;
    performLogIn(input.value);
  };

  // You can still use onChange to trigger side-effects if needed

  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="password-input">Password:</label>
      <input ref={inputRef} id="password-input" type="password" />
      <button type="submit">
        Submit
      </button>
    </form>
  );
}
```

Key differences & impact

How to consider going one way or the other

Key differences & impact

How to consider going one way or the other

- Philosophy.
- Consistency.
- **Feature complexity.**
- **Code complexity.**
- **Performance.**

Key takeaways

If there’s only one slide you should refer to, it’s that one (the next one)

Key takeaways

If there's only one slide you should refer to, it's that one

- Both approaches are **fine**.
- There is **no React police**.
- Use whatever **works best for you** and your use case.
- Consider the implications, alternatives & **keep an open mind**.
- Remember the **rule of least power**.