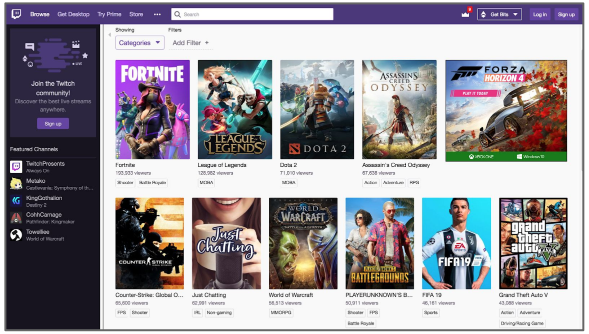
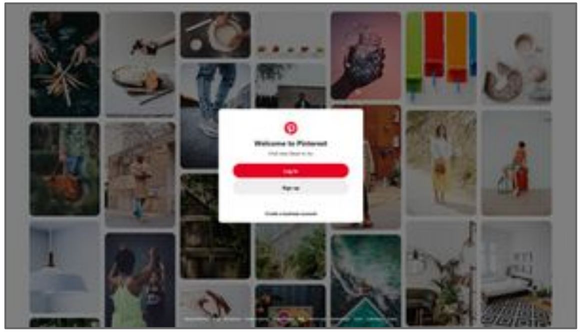
Forms and Controlled Components

Course: Java

S1

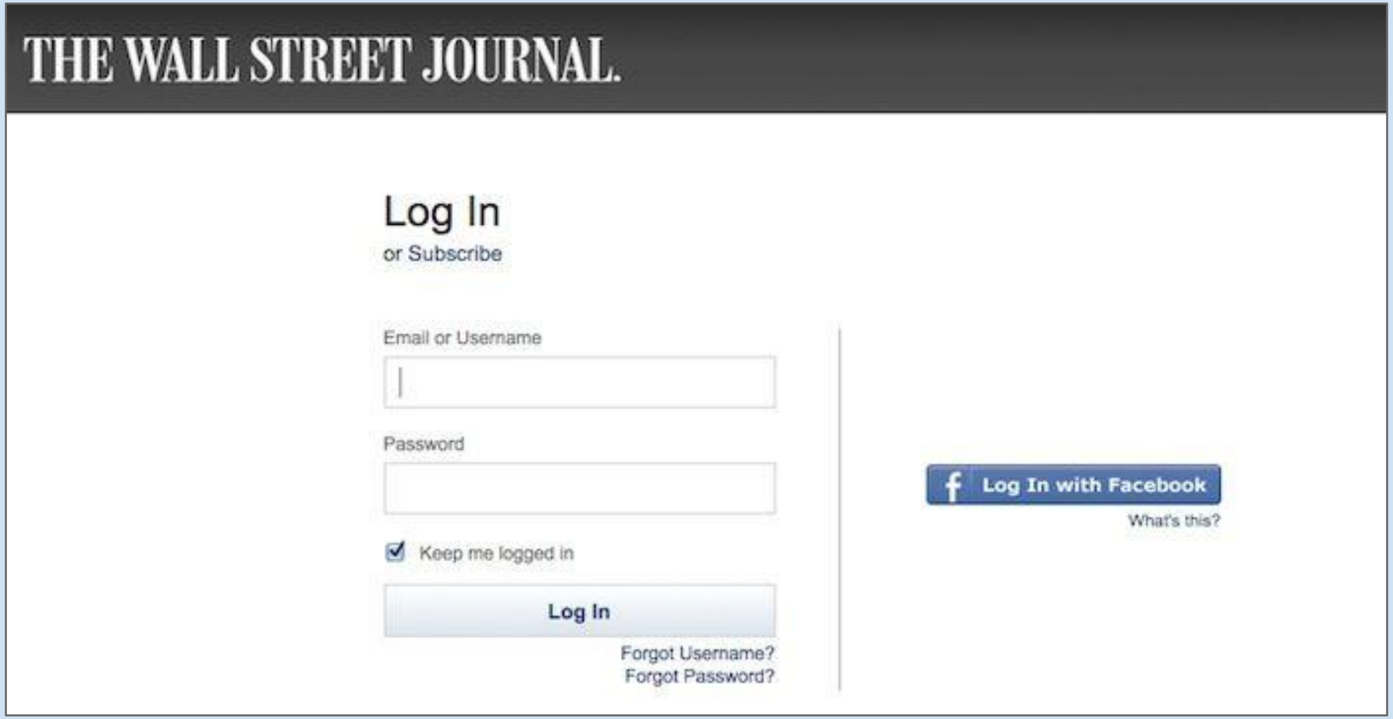
HTML Forms in React

HTML forms turned the web into an application-hosting environment. Without them, many of the sites that we use every day couldn't exist. These include Google search, Baidu, YouTube, Twitch, Pinterest, etc.

HTML Forms in React (continued)

Even applications that are relatively read-only, like the *Wall Street Journal*, require a login form to access protected read-only content or use form controls to drive UI interaction.


THE WALL STREET JOURNAL.

Log In

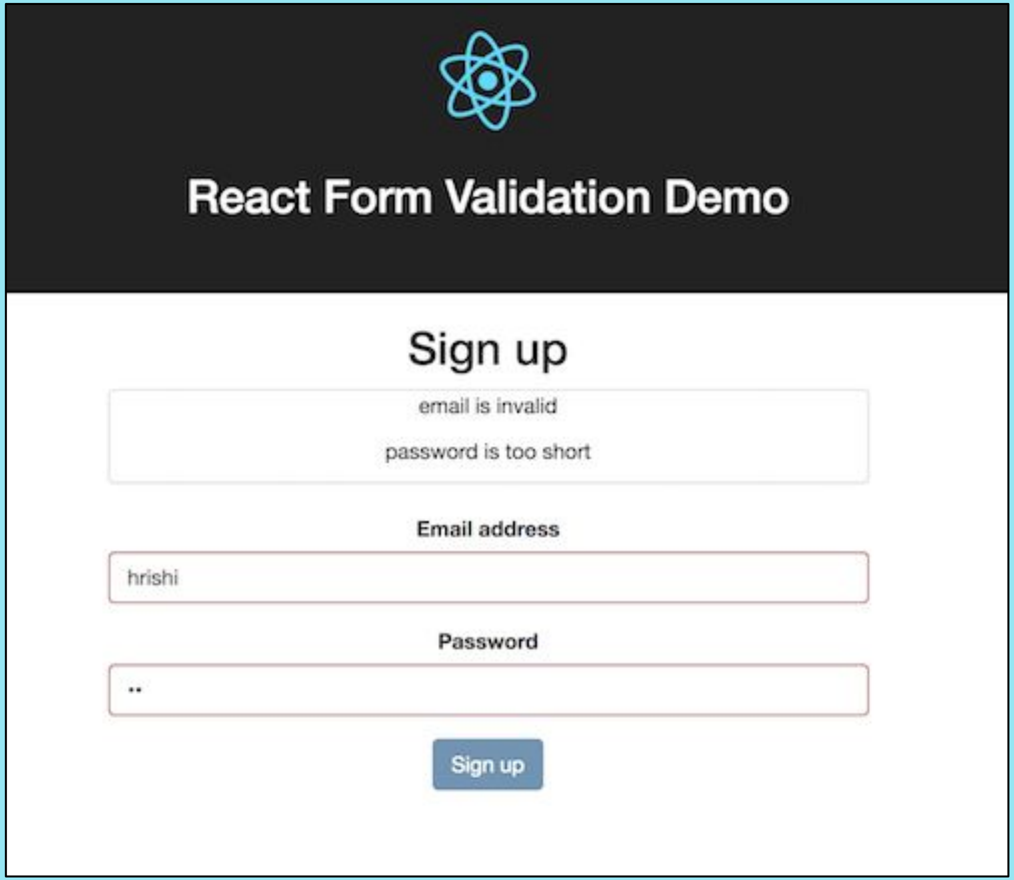
or Subscribe

Email or Username

Password

☑ Keep me logged in

React has its own way of handling HTML forms.



email is invalid

password is too short

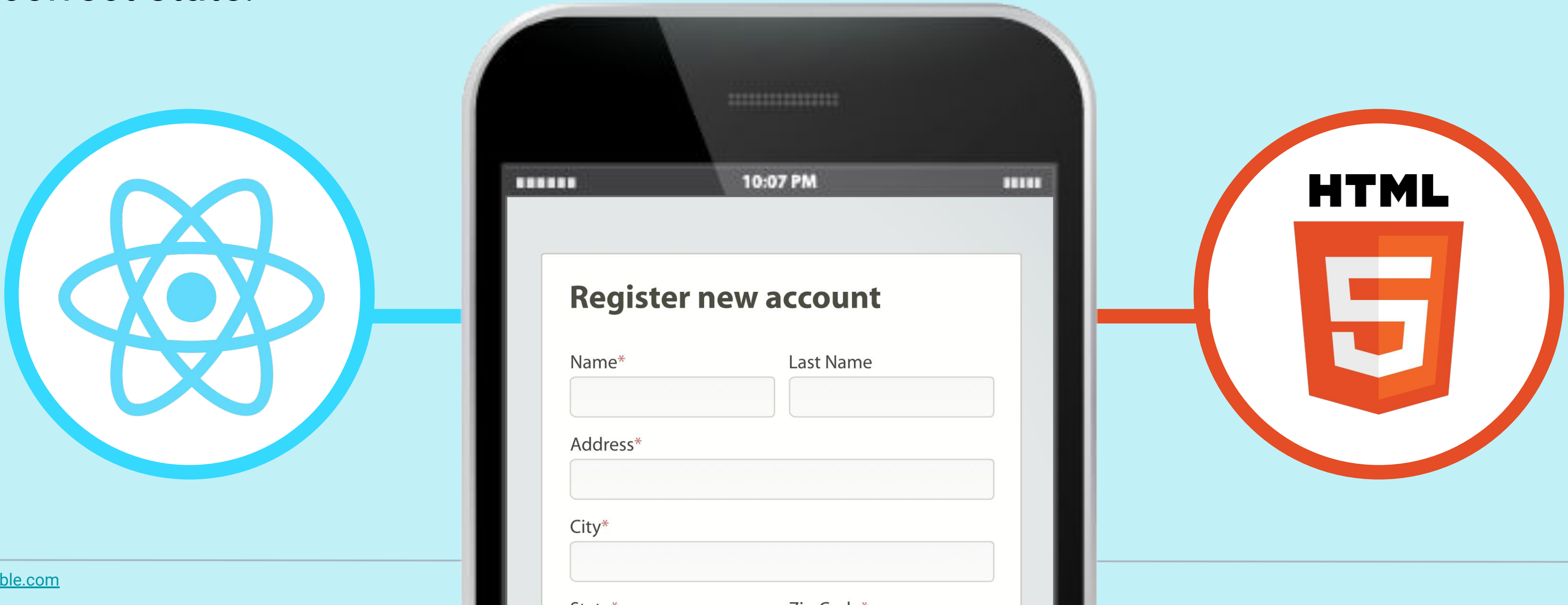
Email address: hrishi

Password: ••

HTML Forms in React (continued)

Controlled components are form controls that are synchronized with React state.

React strives to have a single representation of state, so it recommends that form controls never reflect a different value than what is considered the current, correct state.



HTML Forms in React (continued)

Controlled components are form controls that are synchronized with React state.

React strives to have a single representation of state, so it recommends that form controls never reflect a different value than what is considered the current, correct state.

Controlled components	Uncontrolled components
They do not maintain their own state.	They maintain their own state.
Data is controlled by the parent component.	Data is controlled by the DOM.
They take in the current values through props and then notify the changes via callbacks.	Reefs are used to get their current values.

HTML Forms in React (continued)

We can use a lot of what we already know about HTML forms and form controls to be effective in React.

React adds an extra layer of caution on top.

Learning Outcomes

By the end of this lesson, you will be able to:

01 Use controlled components.

02 Write a generic `onChange` handler that updates all properties in a tracked object.

03 Handle form submission.

04 Create a form with React Bootstrap.

05 Validate a form.

Forms

Many third-party tools are available for form management.

We’ll stick with the basics to lay a good foundation.

Controlled components are an HTML control synchronized with state. The control's value and state agree at all times.

Controlled Components

01

React state

02

Form control value set with state

03

An onChange handler to set state

```
const [subject, setSubject] = useState("");

return (
    <>
        <h1>Contact Us</h1>
        <form>
            <div className="mb-3">
                <label htmlFor="subject">Subject</label>
                <input type="text" id="subject" className="form-control"
                    value={subject} onChange={(evt) => setSubject(evt.target.value)} />
            </div>
        </form>
        <div>{subject}</div>
    </>
);
```

Multiple Controlled Components

We will never manage individual properties as state. A form should always map to an object. To add more controlled components, add state and form controls.

```
const [name, setName] = useState("");
const [subject, setSubject] = useState("");

return (
    <form>
        <div className="mb-3">
                <label htmlFor="name">Name</label>
                <input type="text" id="name" className="form-control"
                    value={name} onChange={(evt) => seetName(evt.target.value)} />
            </div>
        <div className="mb-3">
            <label htmlFor="subject">Subject</label>
            <input type="text" id="subject" className="form-control"
                value={subject} onChange={(evt) => setSubject(evt.target.value)} />
        </div>
    </form>
);
```

Track an Object

A full object is bound to a form. As form elements change, the object is updated and is always current. There's no need to assemble a bunch of control values on form submission. The tracked object already contains the control values.

In the generic `handleChange` function, the control's name (`evt.target.name`) is a better choice than its id. The name works for multielements like radio button or checkbox groups.

It's not common to track individual pieces of state. Components prefer working with objects.

With an object, we need a new strategy for handling changes.

We can use a single `onChange` handler if we're careful with naming conventions.

```
const [message, setMessage] = useState({
    name: "",
    subject: ""
});

const handleChange = (evt) => {
    const clone = { ...message };
    clone[evt.target.name] = evt.target.value;
    setMessage(clone);
    console.log(clone);
};

return (
    <form>
        <div className="mb-3">
            <label htmlFor="name">Name</label>
            <input type="text" id="name" name="name"
                className="form-control"
                value={message.name} onChange={handleChange} />
        </div>
        <div className="mb-3">
            <label htmlFor="subject">Subject</label>
            <input type="text" id="subject" name="subject"
                className="form-control"
                value={message.subject} onChange={handleChange} />
        </div>
    </form>
);
```

<textarea>

Different form controls require slightly different handling.

A `<textarea>` typically embeds content between its tags. As a controlled component, we set its value like a text input.

Our `handleChange` function works without edits.

```
// state
const [message, setMessage] = useState({
    name: "",
    subject: "",
    body: ""
});

// JSX
<div className="mb-3">
    <label htmlFor="body">Message</label>
    <textarea id="body" name="body" className="form-control"
        value={message.body} onChange={handleChange}></textarea>
</div>
```

Each control type is slightly different, but the pattern is the same:

- Set a value with state.
- Update state on change.
- This resets the value.

<select>

Encourage an initial value for `select`s to have something selected when the user arrives. If the `select` should start empty, consider an empty option:

`<option value="">Please select something.</option>`

- The `<select>` element follows the same pattern.
- Set its value. If an option value matches, it's selected.
- The `handleChange` function still works.

```
// state
const [message, setMessage] = useState({
    name: "",
    subject: "",
    body: "",
    reason: "feedback"
});

// JSX
<label htmlFor="reason">Reason</label>
<select id="reason" name="reason" className="form-control"
    value={message.reason} onChange={handleChange}>
    <option value="service">Request Service</option>
    <option value="issue">Report an Issue</option>
    <option value="sales">Sales Inquiry</option>
    <option value="feedback">General Feedback</option>
</select>
```

input type="radio"

Checkable controls like radio and checkbox use the `checked` attribute instead of `value`.

Express a Boolean condition. A `true` value checks the control.

`handleChange` is still valid.

```
// state
const [message, setMessage] = useState({
    name: "",
    subject: "",
    body: "",
    reason: "feedback",
    bestContact: ""
});

// JSX
<div className="col">
    <input type="radio" value="morning" id="morning" name="bestContact"
        checked={message.bestContact === "morning"} onChange={handleChange} />
    <label htmlFor="morning"> Morning</label>
</div>
<div className="col">
    <input type="radio" value="afternoon" id="afternoon" name="bestContact"
        checked={message.bestContact === "afternoon"} onChange={handleChange} />
    <label htmlFor="afternoon"> Afternoon</label>
</div>
<div className="col">
    <input type="radio" value="evening" id="evening" name="bestContact"
        checked={message.bestContact === "evening"} onChange={handleChange} />
    <label htmlFor="evening"> Evening</label>
</div>
```

input type="checkbox"

We could assign a value “true” to a checkbox, but we don't get a Boolean—“true” is still a string. To set a Boolean property, we have to inspect checked.

- A checkbox is checkable, so we use the checked attribute.
- Unfortunately, our handleChange function is now broken.
- We want a Boolean for the acceptedTerms property. A checkbox’s value is a string—by default, “on”.

```
// state
const [message, setMessage] = useState({
    name: "",
    subject: "",
    body: "",
    reason: "feedback"
    bestContact: "",
    acceptedTerms: false

});

// JSX
<div className="form-check">
    <input className="form-check-input" type="checkbox"
        id="acceptedTerms" name="acceptedTerms"
        checked={message.acceptedTerms} onChange={handleChange} />
    <label className="form-check-label" htmlFor="acceptedTerms">
        I've read and accept the terms and conditions.
    </label>
</div>
```

Fix handleChange

To keep handleChange universal, check the control type.

If it's a checkbox, use the checked attribute.

Otherwise, use value.

Depending on our UI, other elements may require special handling as well: checkbox lists, multiselects, etc.

```
const handleChange = (evt) => {

    // Check control type.
    const value = evt.target.type === "checkbox" ?
        evt.target.checked : evt.target.value;

    const clone = { ...message };
    clone[evt.target.name] = value;
    setMessage(clone);

    // Debugging...
    console.log(clone);
};
```

Form Submission

Forms will send a full GET or POST and reload the view if not stopped from submission. If you ever notice a page being reset on submission, troubleshoot the form first.

- Standard form submission reloads the view. We don't want that in React. Prevent the event with `preventDefault()`.
- Always handle `onSubmit`, never a submit button's `onClick`.
- Often, submit collaborates with a parent component.

```
const handleChange = (evt) => {
    // No postback form submission.
    evt.preventDefault();

    // Do something with message.
    // It's always up-to-date.
    console.log(message);

    // Often, we'd send our object up
    // to a parent.
    // parentCallback(message);
};

<form onSubmit={handleSubmit}>
    {/* Lots more JSX */}
</form>
```

Break

Time to Code

Form-Driven ToDo Walkthrough

Suggested Time:

15 Minutes

Questions?

React Bootstrap Forms

You will need to learn the React Bootstrap API on your own time.

React Bootstrap Forms

- Every Bootstrap “component” has a corresponding React Bootstrap React component.
- Forms are no different.
- React Bootstrap components accept HTML attributes,
  so it’s possible to convert with identical behavior.

```
<Form onSubmit={handleSubmit}>
    <Form.Group controlId="name">
        <Form.Label>Name</Form.Label>
        <Form.Control type="text" name="name"
            value={message.name} onChange={handleChange} />
    </Form.Group>
    <Form.Group controlId="subject">
        <Form.Label>Subject</Form.Label>
        <Form.Control type="text" name="subject"
            value={message.subject} onChange={handleChange} />
    </Form.Group>
    {/* More JSX */}
</Form>
```

React Bootstrap Form Validation

A brief overview of form validation:

- React Bootstrap includes Bootstrap's form validation classes.
- To enable validation, track whether the form has been validated with Boolean state.
- Add validation checks to the submit handlers.

```
const [validated, setValidated] = useState(false);

const handleSubmit = (evt) => {
    evt.preventDefault();

    const form = evt.currentTarget;
    if (form.checkValidity() === false) {
        // 1. Handle validation failure.
    }

    setValidated(true);

    // 2. Do something with message.
};
```

Time to Code

React Bootstrap Validation Walkthrough

Suggested Time:

10 Minutes

Questions?

Activity: HTML Forms in React

Suggested Time:

30 Minutes

Time’s Up! Let’s Review.

Questions?

RECAP

Learning Outcomes

By the end of this lesson, you will be able to:

01 Use controlled components.

02 Write a generic `onChange` handler that updates all properties in a tracked object.

03 Handle form submission.

04 Create a form with React Bootstrap.

05 Validate a form.

The End