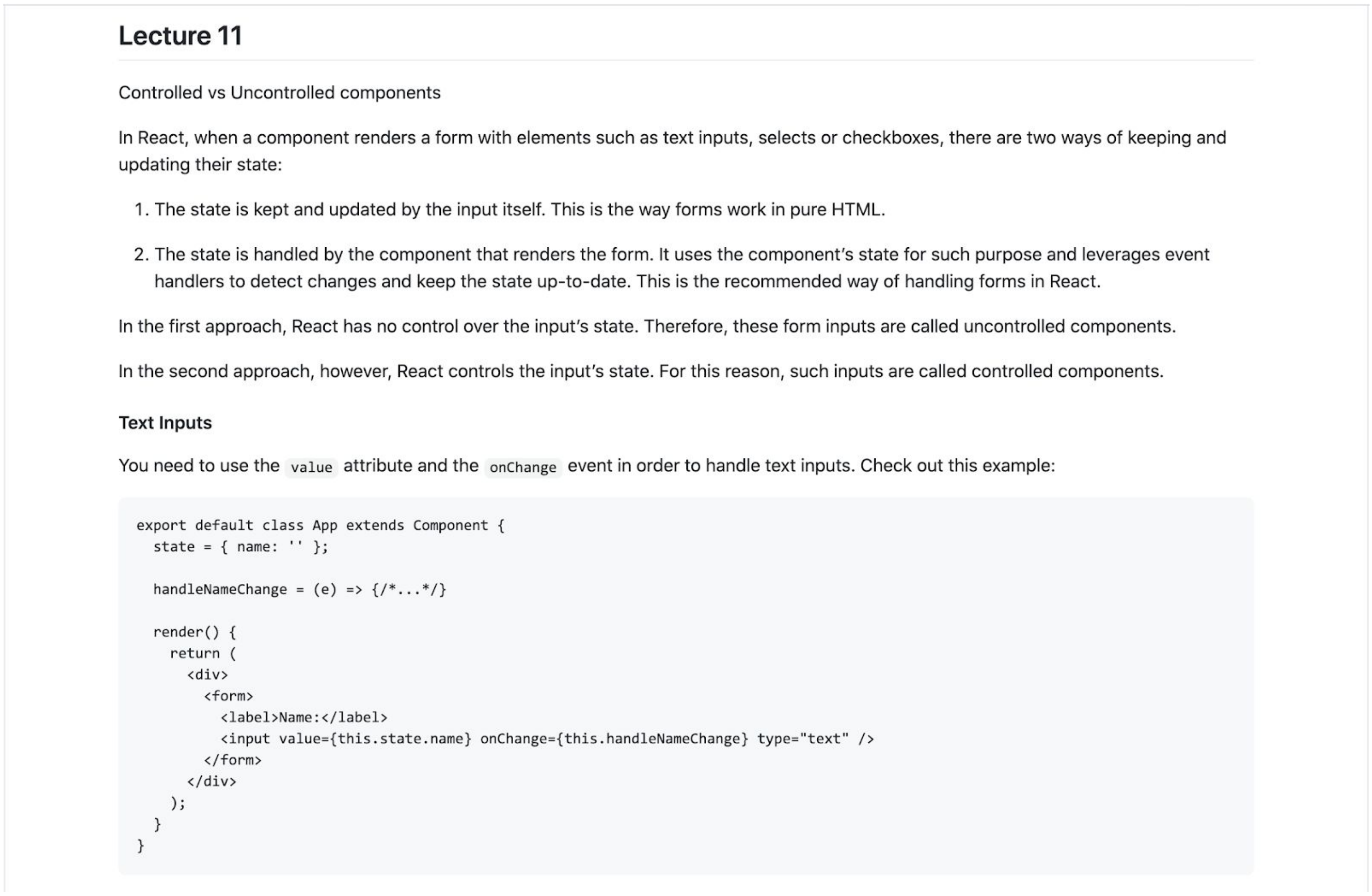
Date : 10 - 07 - 2020
Morning Session : 9 am – 11.00 PM
By ~ Sundeep Charan Ramkumar Today

Topics: React - Day 11 (Project - Part 1)



Lecture 11

Controlled vs Uncontrolled components

In React, when a component renders a form with elements such as text inputs, selects or checkboxes, there are two ways of keeping and updating their state:

1. The state is kept and updated by the input itself. This is the way forms work in pure HTML.
2. The state is handled by the component that renders the form. It uses the component's state for such purpose and leverages event handlers to detect changes and keep the state up-to-date. This is the recommended way of handling forms in React.

In the first approach, React has no control over the input's state. Therefore, these form inputs are called uncontrolled components.

In the second approach, however, React controls the input's state. For this reason, such inputs are called controlled components.

Text Inputs

You need to use the `value` attribute and the `onChange` event in order to handle text inputs. Check out this example:

```
export default class App extends Component {
  state = { name: '' };

  handleNameChange = (e) => {/*...*/}

  render() {
    return (
      <div>
        <form>
          <label>Name:</label>
          <input value={this.state.name} onChange={this.handleNameChange} type="text" />
        </form>
      </div>
    );
  }
}
```

In the above component, every time a change occurs in our input, the handleNameChange event handler is called. Let's take a look at it:

```
handleNameChange = (e) => {
  const name = e.target.value;
  this.setState({ name });
};
```

This handler receives an event object that holds, among other things, the up-to-date value of the input. After getting it, this same value is assigned to its matching entry in the component's state by using the `setState` method.

After the change is performed, the value attribute of the input is automatically updated, since the this.state.name state entry is bound to it.

Textareas

In pure HTML, a textarea has no value attribute. Its value is defined by its content. In React, however, textareas and common text inputs are handled in a very similar way:

```
export default class App extends Component {
  state = { bio: '' };

  handleBioChange = (e) => {
    const bio = e.target.value;
    this.setState({ bio });
  };

  render() {
    return (
      <form>
        <label>Bio:</label>
        <textarea value={this.state.bio} onChange={this.handleBioChange} />
      </form>
    );
  }
}
```

As you may have noticed, the above code is almost identical to the text input example. In both, we use the onChange event and the value attribute to keep the component's and input's states synchronized.

Selects

Similarly to textareas, select tags also have no value attribute in HTML. However, their children – option tags – do have it. To set the value of a select tag, you'd need to add a selected attribute to one of its options. React favors a simpler approach, by bringing the simplicity of the value attribute to select tags:

```
render() {
  return (
    <form>
      <label>Desired color:</label>
      <select value={this.state.color} onChange={this.handleColorChange}>
        <option></option>
        <option value="blue">Blue</option>
        <option value="red">Red</option>
        <option value="green">Green</option>
        <option value="yellow">Yellow</option>
      </select>
    </form>
  );
}
```

The handleColorChange event handler has nothing new:

```
handleColorChange = (e) => {
  const color = e.target.value;
  this.setState({ color });
};
```

If you want your dropdown to have multiple values selected, set the multiple attribute to true and assign an array as its value, like this:

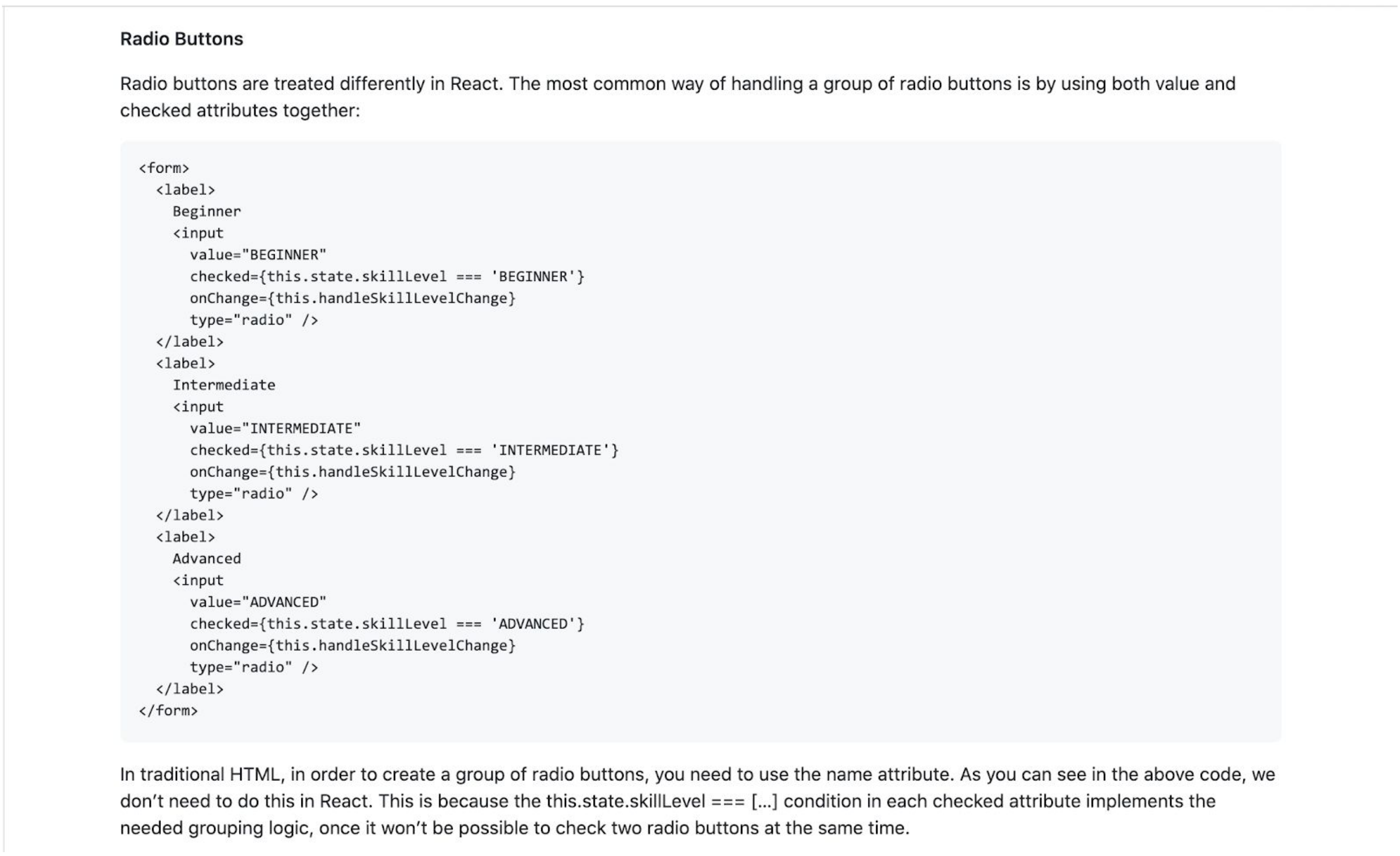
```
render() {
  return (
    <form>
      <label>Desired color:</label>
      <select multiple={true} value={['blue', 'red']}>
        <option></option>
        <option value="blue">Blue</option>
        <option value="red">Red</option>
        <option value="green">Green</option>
        <option value="yellow">Yellow</option>
      </select>
    </form>
  );
}
```

Checkboxes

Checkboxes use the checked attribute instead of the value one. Both work in a very similar manner, but the value of the checked attribute must be a boolean. Check out this example:

Checkboxes use the checked attribute instead of the value one. Both work in a very similar manner, but the value of the checked attribute must be a boolean. Check out this example:

```
state = { acceptedAgreement: false };

handleAcceptAgreementChange = (e) => {
  const acceptedAgreement = e.target.checked;
  this.setState({ acceptedAgreement });
};

render() {
  return (
    <form>
      <label>
        I accept the agreement
        <input
          checked={this.state.acceptedAgreement}
          onChange={this.handleAcceptAgreementChange}
          type="checkbox" />
      </label>
    </form>
  );
}
```

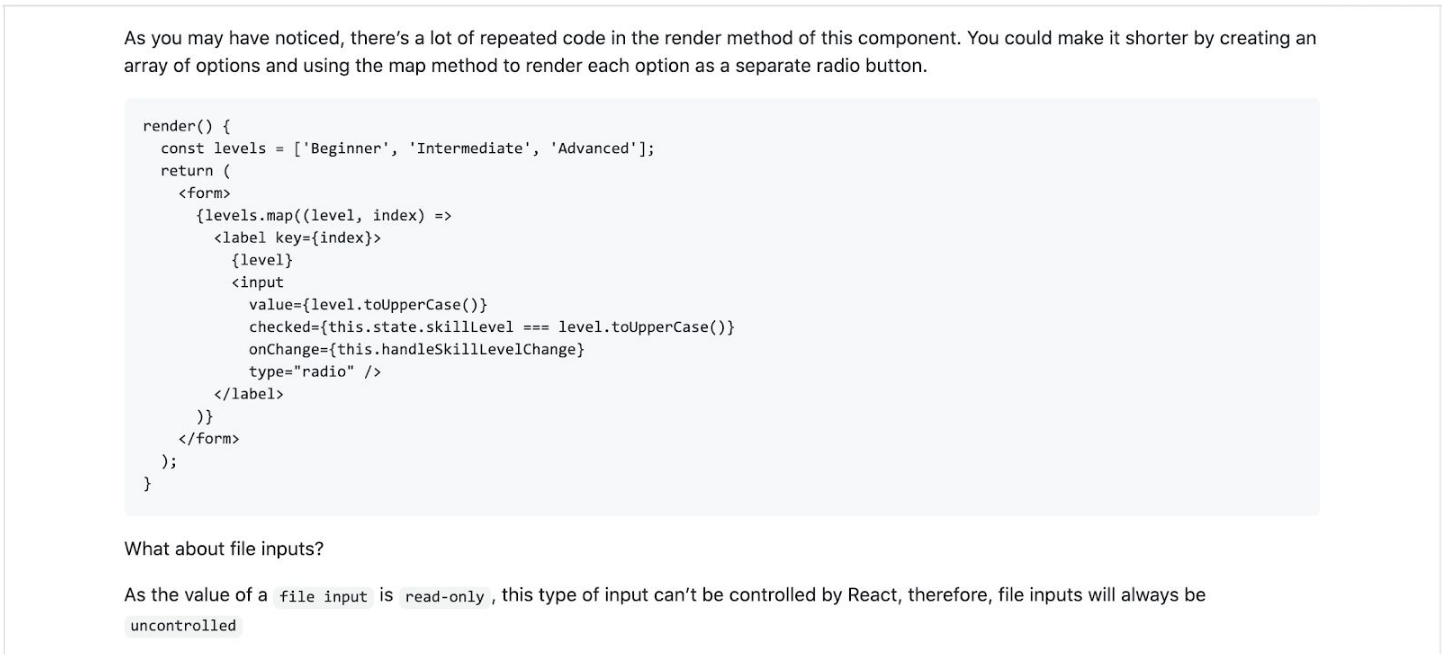


Radio Buttons

Radio buttons are treated differently in React. The most common way of handling a group of radio buttons is by using both value and checked attributes together:

```
<form>
  <label>
    Beginner
    <input
      value="BEGINNER"
      checked={this.state.skillLevel === 'BEGINNER'}
      onChange={this.handleSkillLevelChange}
      type="radio" />
  </label>
  <label>
    Intermediate
    <input
      value="INTERMEDIATE"
      checked={this.state.skillLevel === 'INTERMEDIATE'}
      onChange={this.handleSkillLevelChange}
      type="radio" />
  </label>
  <label>
    Advanced
    <input
      value="ADVANCED"
      checked={this.state.skillLevel === 'ADVANCED'}
      onChange={this.handleSkillLevelChange}
      type="radio" />
  </label>
</form>
```

In traditional HTML, in order to create a group of radio buttons, you need to use the name attribute. As you can see in the above code, we don't need to do this in React. This is because the this.state.skillLevel === [...] condition in each checked attribute implements the needed grouping logic, once it won't be possible to check two radio buttons at the same time.



As you may have noticed, there's a lot of repeated code in the render method of this component. You could make it shorter by creating an array of options and using the map method to render each option as a separate radio button.

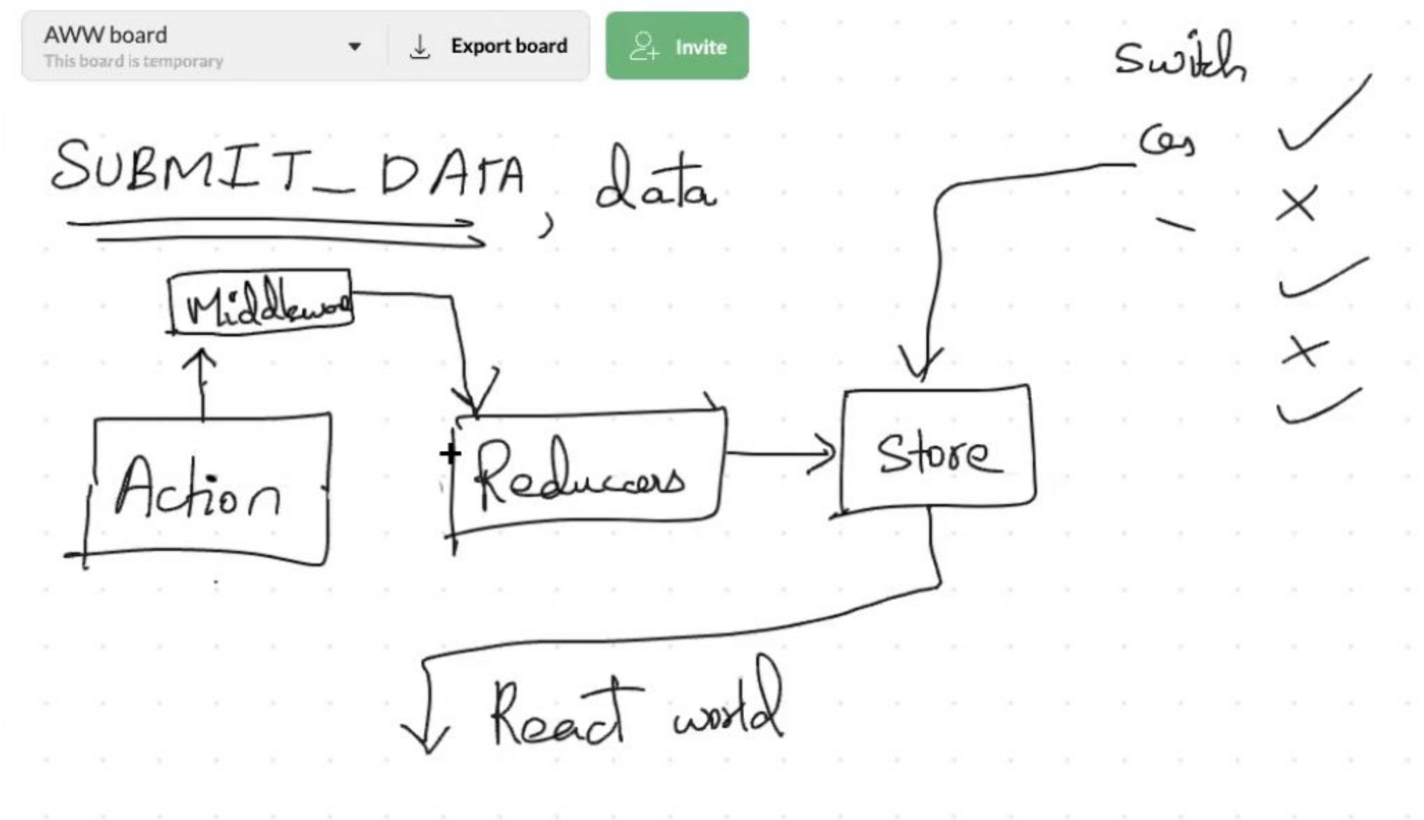
```
render() {
  const levels = ['Beginner', 'Intermediate', 'Advanced'];
  return (
    <form>
      {levels.map((level, index) =>
        <label key={index}>
          {level}
          <input
            value={level.toUpperCase()}
            checked={this.state.skillLevel === level.toUpperCase()}
            onChange={this.handleSkillLevelChange}
            type="radio" />
        </label>
      )}
    </form>
  );
}
```

What about file inputs?

As the value of a `file input` is `read-only`, this type of input can't be controlled by React, therefore, file inputs will always be `uncontrolled`

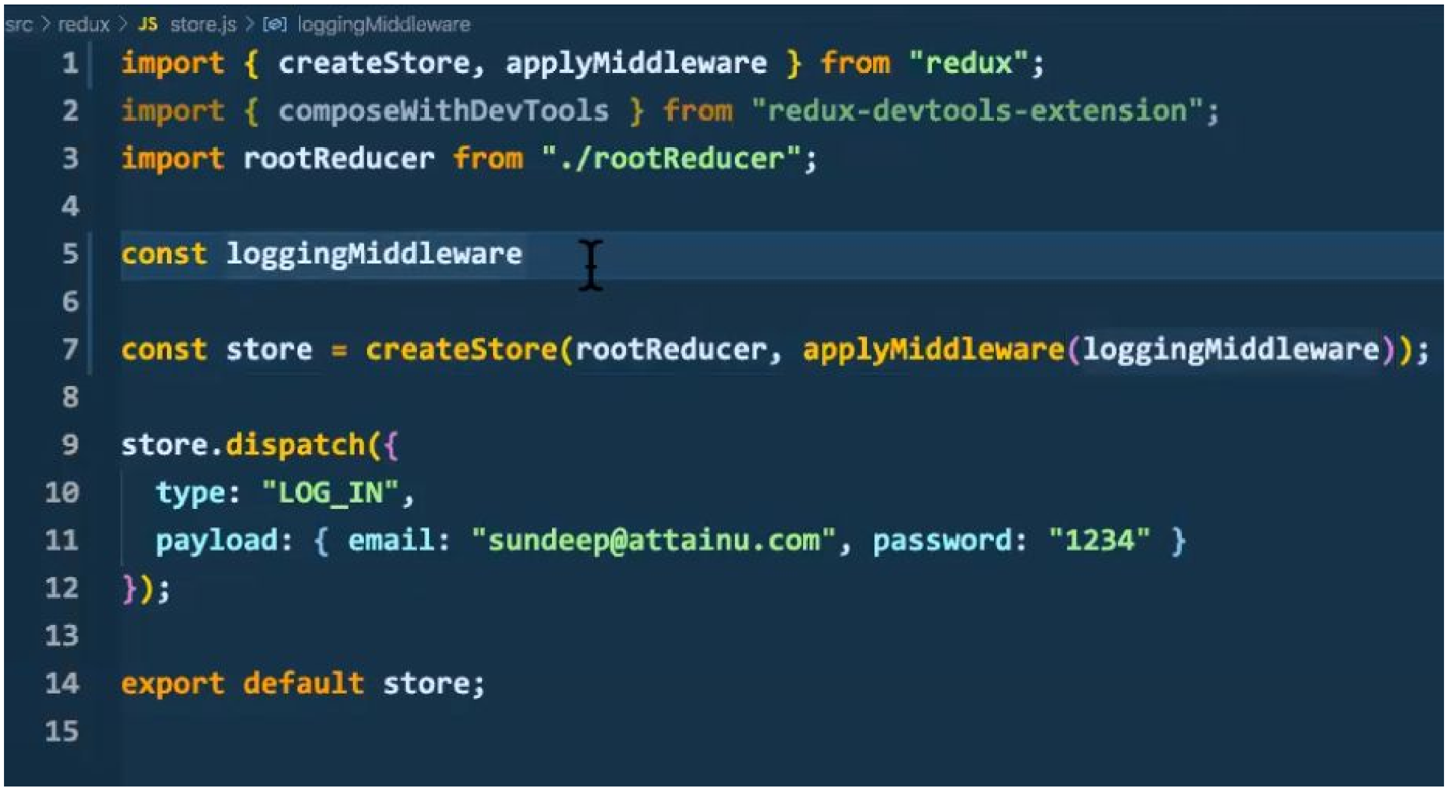
Middleware: is something which allows you to change a configuration for a particular thing.in our front end application in terms of Redux you can specifically mention certain customization to your application .

Where exactly is middleware injected?



https://medium.com/netscape/creating-custom-middleware-in-react-redux-961570459ecb

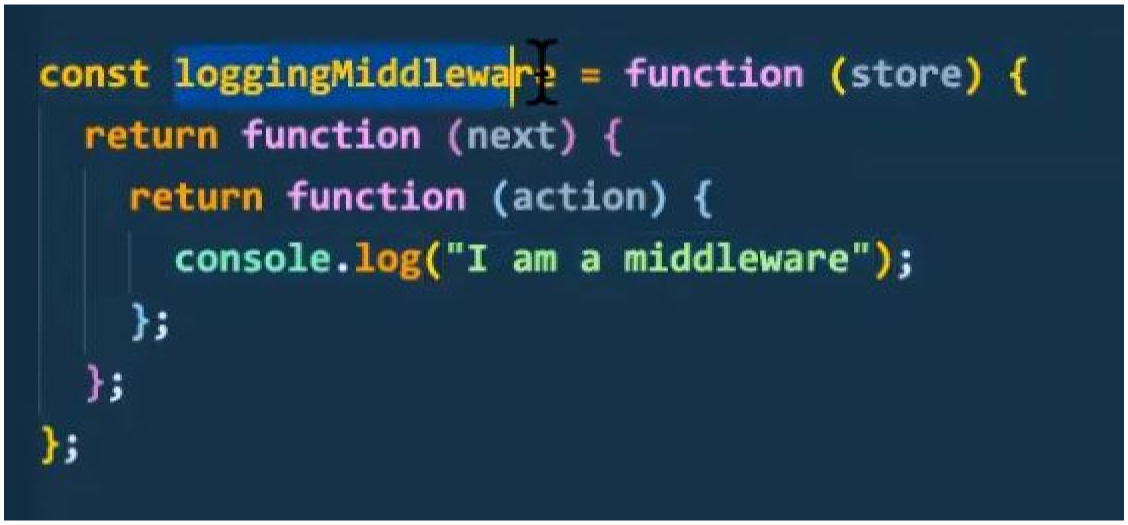
Store.js →



src > redux > store.js > loggingMiddleware

```javascript
import { createStore, applyMiddleware } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";
import rootReducer from "./rootReducer";

const loggingMiddleware

const store = createStore(rootReducer, applyMiddleware(loggingMiddleware));

store.dispatch({
  type: "LOG_IN",
  payload: { email: "sundeep@attainu.com", password: "1234" }
});

export default store;

```

Structure of middleware: **middleware is going to be a function which returns a function and that in a function will return another function.**

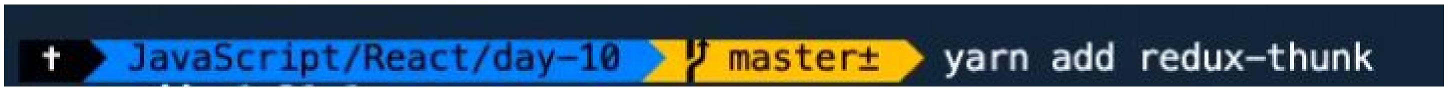


```
const loggingMiddleware = function (store) {
  return function (next) {
    return function (action) {
      console.log("I am a middleware");
    };
  };
};
```

Redux Thunk:

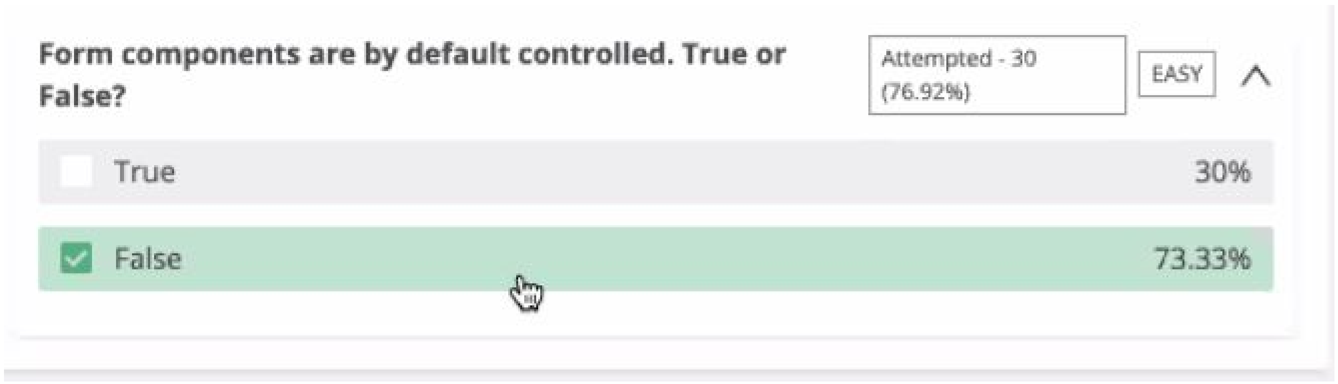
Redux thunk allows you to make asynchronous requests and passing dispatches.

Installing redux Thunk:



```
† JavaScript/React/day-10  master± yarn add redux-thunk
```

https://medium.com/@ingridf/what-redux-thunk-is-how-to-install-it-and-how-it-works-b67b815d2672

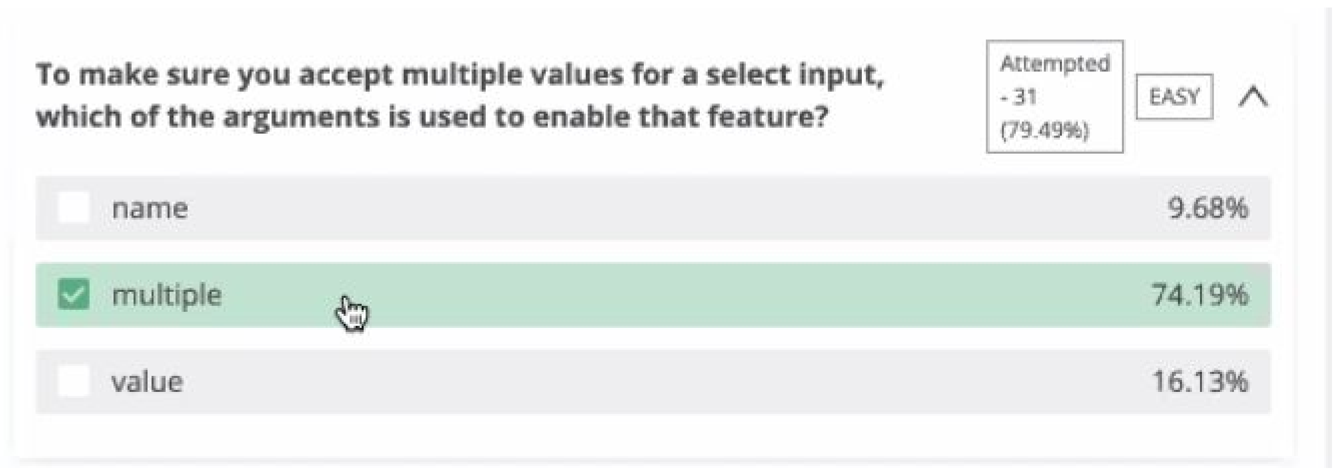


Form components are by default controlled. True or False?

Attempted - 30 (76.92%)

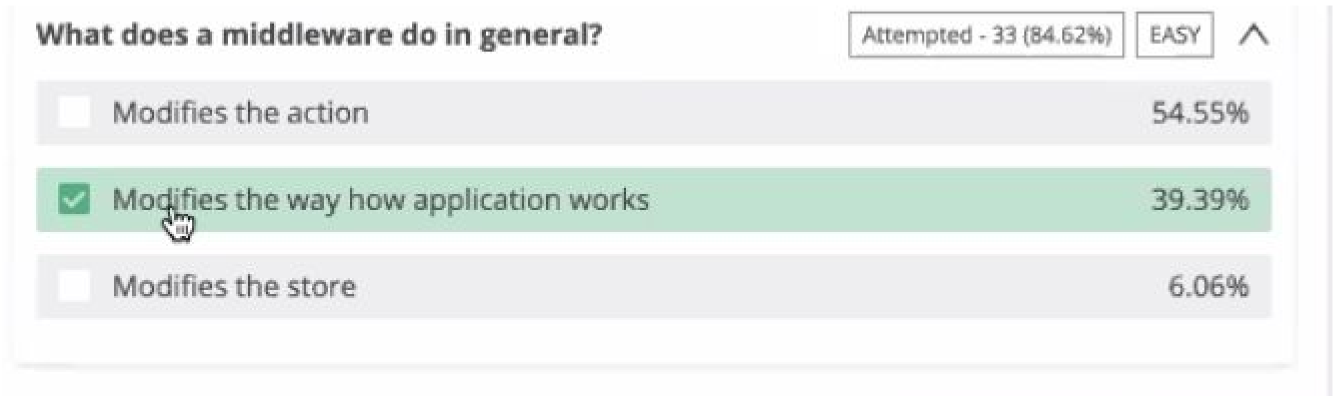
EASY

☐ True	30%
✓ False	73.33%


To make sure you accept multiple values for a select input, which of the arguments is used to enable that feature?

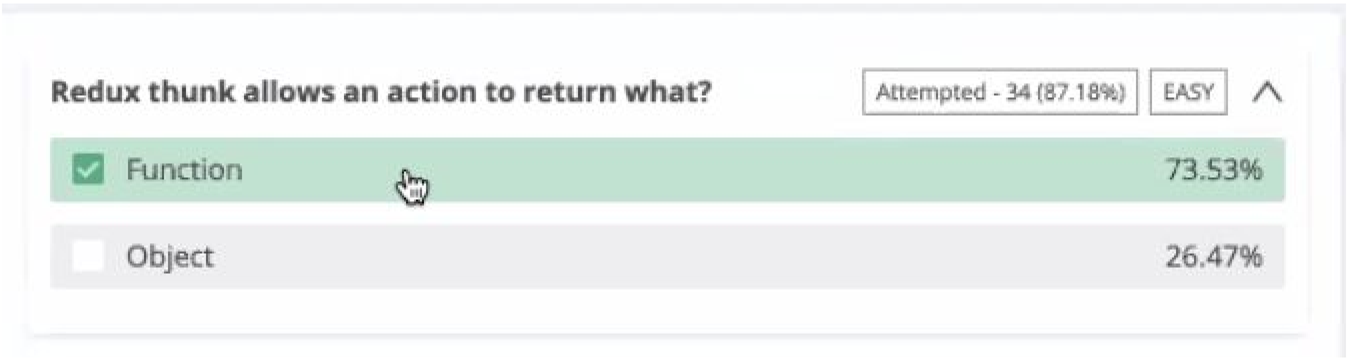
Attempted - 31 (79.49%) EASY

name	9.68%
✓ multiple	74.19%
value	16.13%


What does a middleware do in general?

Attempted - 33 (84.62%) EASY

Modifies the action	54.55%
✓ Modifies the way how application works	39.39%
Modifies the store	6.06%


Redux thunk allows an action to return what?

Attempted - 34 (87.18%) EASY

✓ Function	73.53%
Object	26.47%

Dave CEDDIA, Pure React (Updated till hooks).

https://attainu-ramanujan.slack.com/archives/C010HUNDB3K/p1594611755007000