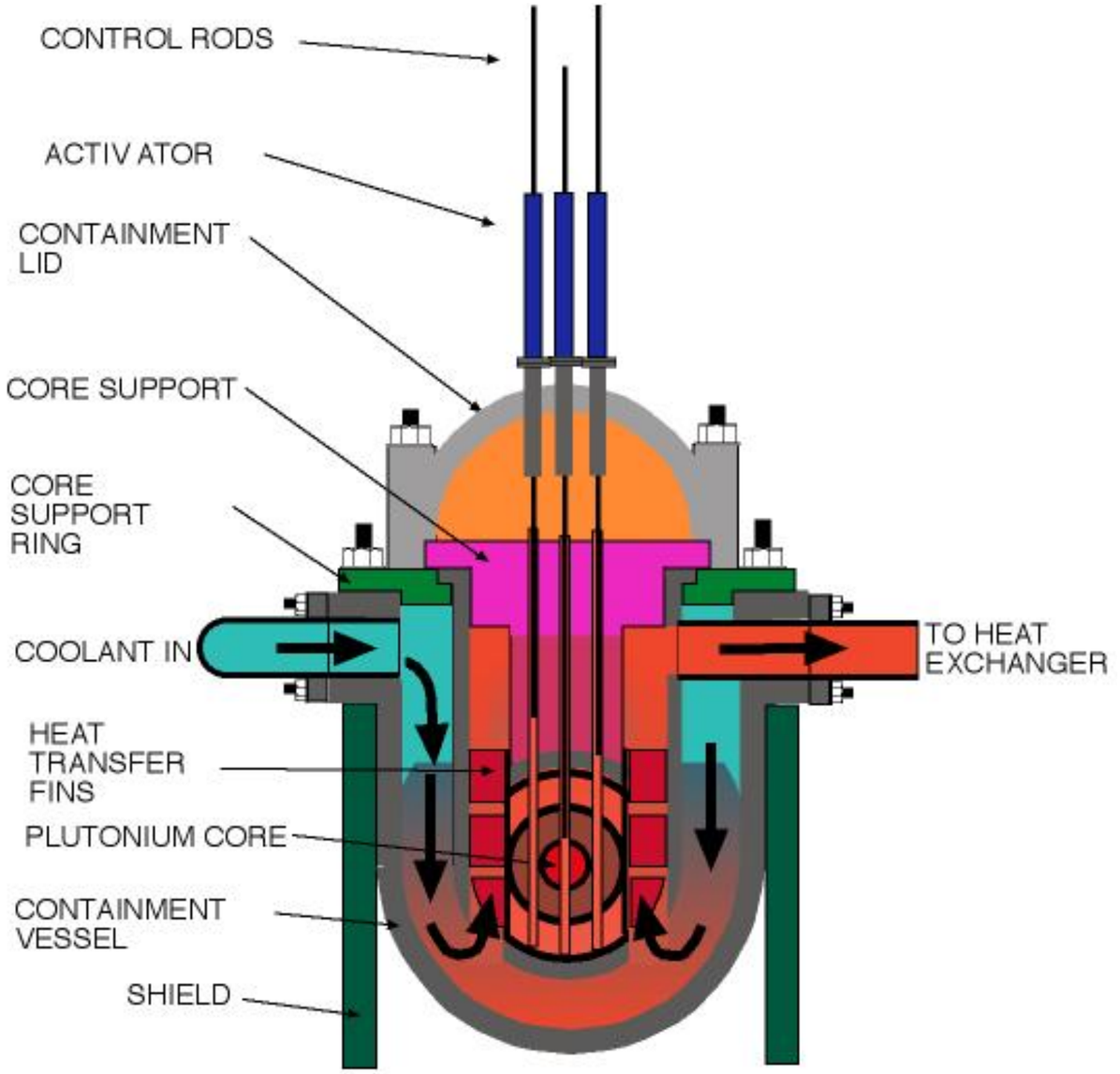
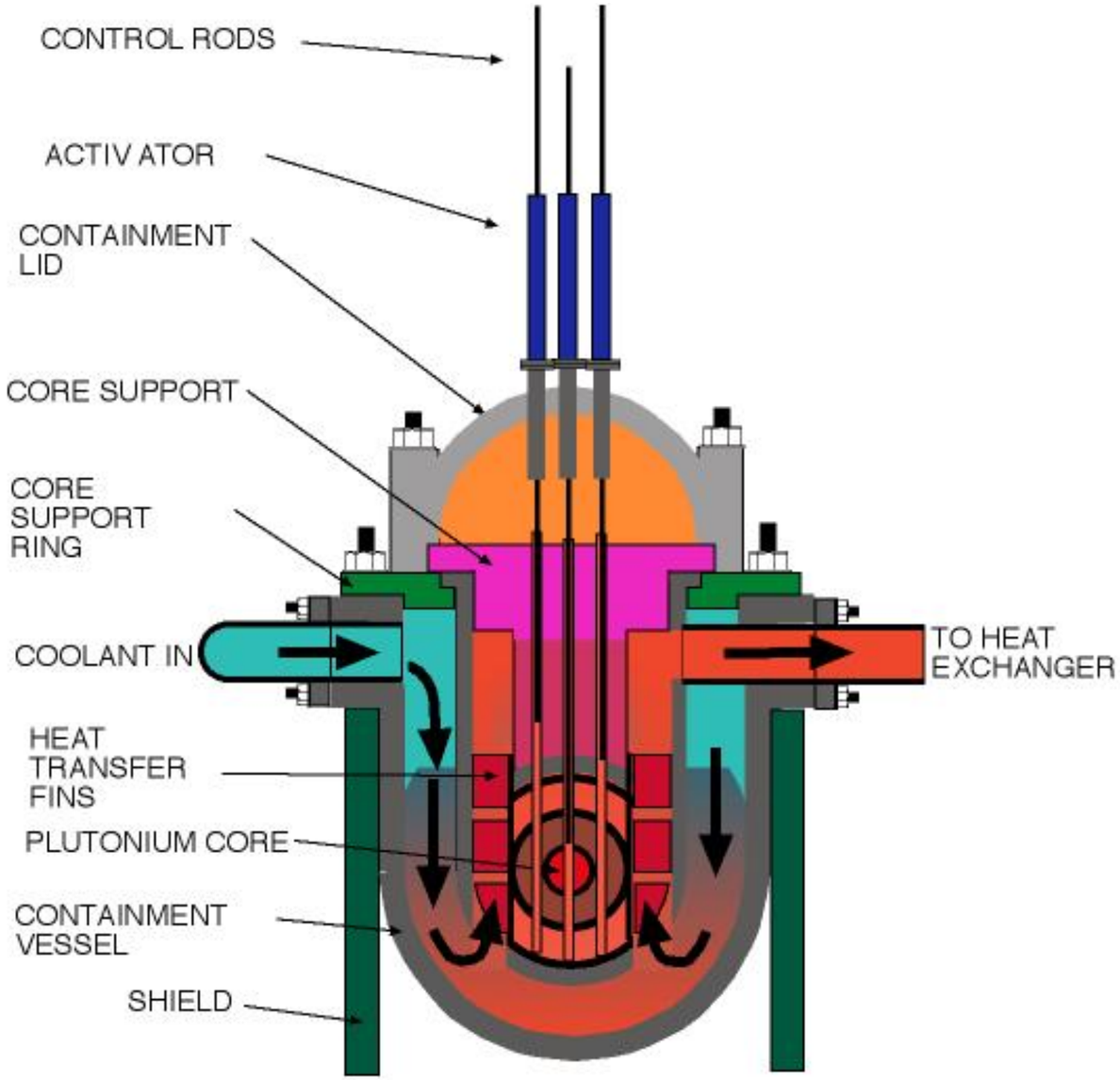
Building submarines that don't leak



SUBMARINE NUCLEAR REACTOR



SUBMARINE NUCLEAR REACTOR

setState?

Flux?

findDOMNode?

componentWillUnmount?

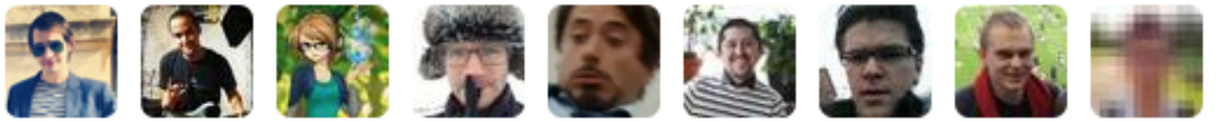
this.props.onChange?

this.props.children?

Horse JS @horse_js · Jul 1

Sure, any large code base needs to be modular, but why

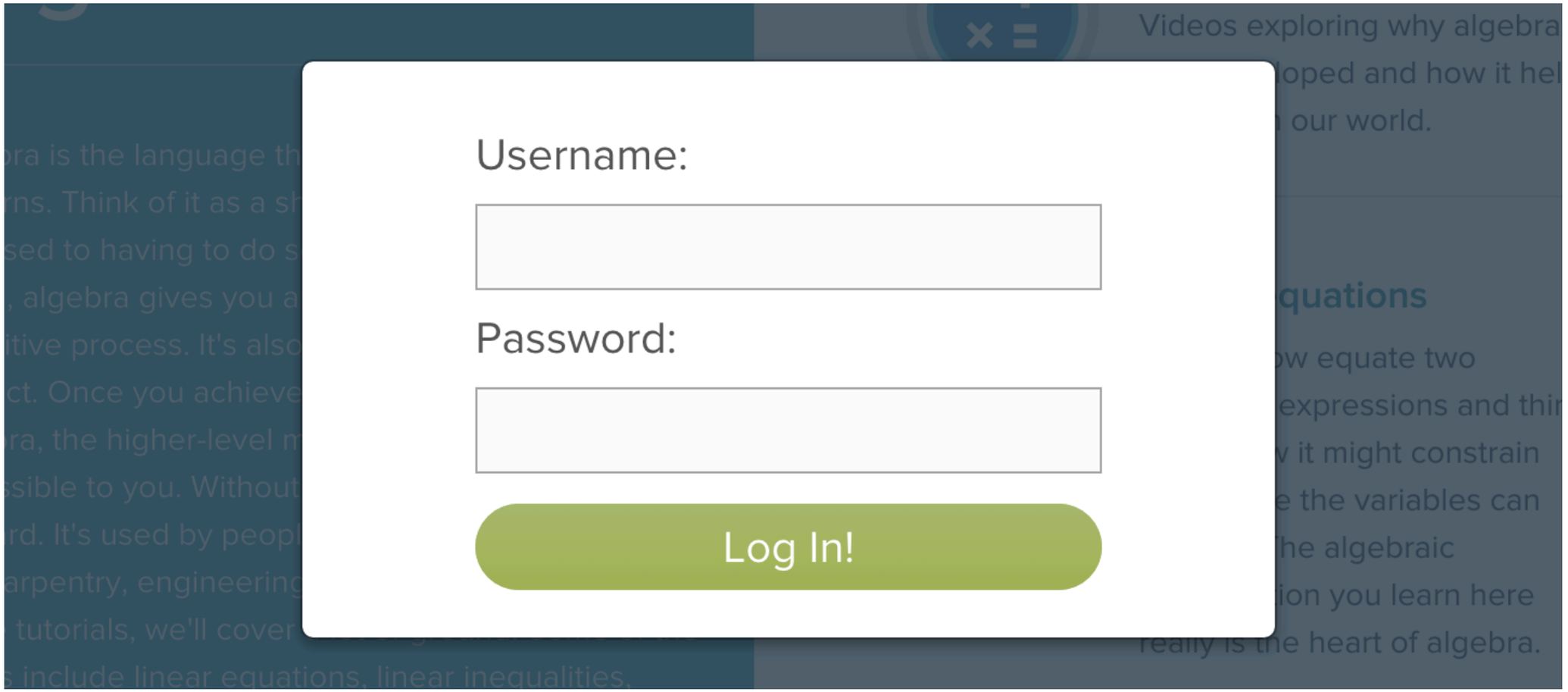
RETWEETS	FAVORITES
11	19

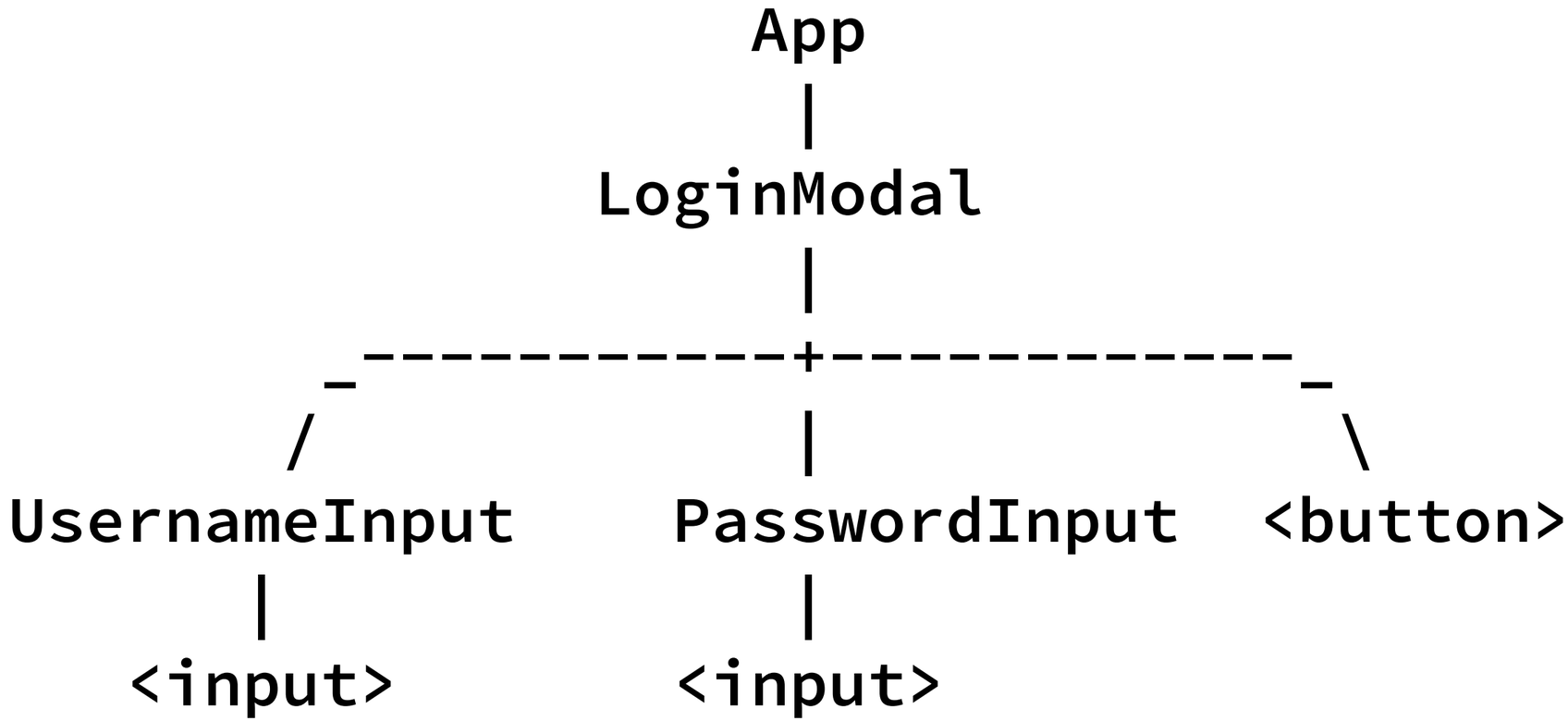


7:45 AM - 1 Jul 2015 · Details

Encapsulating your components

Login Modal




App
LoginModal
UsernameInput
PasswordInput
<button>
<input>
<input>

LoginModal

```
render: function() {
    return <div className="overlay" onClick={this._dismiss}>
        <div className="modal" onClick={this._ignore}>
            <UsernameInput ref="username" />
            <PasswordInput ref="password" />
            <button onClick={this._submit}>Log In!</button>
        </div>
    </div>;
},
```

#1

Wrong lifecycle flow

#1: Wrong lifecycle flow

LoginModal

```
_submit: function() {
    login(
        React.findDOMNode(this.refs.username).value,
        React.findDOMNode(this.refs.password).value,
        function(user) {
            AppActions.registered(user);
        }
    );
},
```

#1: Wrong lifecycle flow: fix

LoginModal

```
_submit: function() {
    login(
        AppStore.getCurrentUsername(),
        AppStore.getCurrentPassword(),
        function(user) {
            AppActions.registered(user);
        }
    );
},
```

#1: Wrong lifecycle flow: fix

`UsernameInput`

```
componentWillMount: function() {
    AppStore.addListener(function(state) {
        this.setState({
            value: state.currentUsername,
        });
    }.bind(this));
},
```

#1: Wrong lifecycle flow: fix

`UsernameInput`

```
<label>
    Username:
    <input type="text" value={this.state.currentUsername}
        onChange={function(e) {
            AppActions.setUsername(e.target.value);
        }} />
</label>
```

#2

OVER-FLUXING

#2: Over-fluxing

* 2 types of components:
    * Know about app world
    * Reusable components
* Minimize your flux surface area

#2: Over-fluxing: fix

LoginModal

```
componentWillMount: function() {
    AppStore.addListener(function(state) {
        this.setState({
            currentUsername: state.currentUsername,
            currentUsername: state.currentPassword,
        });
    }.bind(this));
},
```

#2: Over-fluxing: fix

LoginModal

```
<UsernameInput
    value={this.state.currentUsername}
    onChange={function(newUsername) {
        AppActions.setUsername(newUsername);
    }} />
```

#2: Over-fluxing: fix

UsernameInput

```
<input type="text" value={this.props.value}
    onChange={function(e) {
        this.props.onChange(e.target.value);
    }.bind(this)} />
```

Propful, not stateful, components

#3

Poor division of props, state, and instance vars

#3: props / state / instance variables

PROPS

```
<UsernameInput
    value={this.state.currentUsername}
    onChange={/* ... */} />
```

#3: props / state / instance variables

* Things that are temporary and don't need to be persisted:
  * Whether the modal is visible
  * Current state of the modal inputs

* Hiding complexity from your parent

#3: props / state / instance variables

LoginModal

```
    <UsernameInput
        value={this.state.currentUsername}
        onChange={this._changeUsername} />

_changeUsername: function(newUsername) {
    this.setState({currentUsername: newUsername});
},
```

#3: props / state / instance variables

* Instance variables: non-renderable data (this._myVar)
    * removing event listeners
    * callback cancellations
    * caches (often things from render)
    * timer IDs

#3: props / state / instance variables

App

```
componentWillMount: function() {
  AppStore.addListener(function(state) {
    this.setState({
      user: state.user,
    });
  }.bind(this));
},
```

#3: props / state / instance variables

App

```
componentWillMount: function() {
  this._listenerId = AppStore.addListener(function(state){
    this.setState({
      user: state.user,
    });
  }.bind(this));
},
componentWillUnmount: function() {
  AppStore.removeListener(this._listenerId);
},
```

#4

Doing things at the wrong level of the tree

#4: Wrong level of the tree

LoginModal:

```
<UsernameInput
    value={this.state.currentUsername}
    onChange={this.setState} />
```

UsernameInput:

```
<input type="text" value={this.props.value}
    onChange={function(e) {
        this.props.onChange(
            {currentUsername: e.target.value}
        );
    }.bind(this)} />
```

#4: Wrong level of the tree

```
LoginModal:
        <UsernameInput
            value={this.state.currentUsername}
            onChange={this._changeUsername} />

    _changeUsername: function(newUsername) {
        this.setState({currentUsername: newUsername});
    },

UsernameInput:
        <input type="text" value={this.props.value}
            onChange={function(e) {
                this.props.onChange(
                    e.target.value
                );
            }.bind(this)} />
```

#4: Wrong level of the tree

* The "Username" text inside the UsernameInput
  * "label" props for every type of input
    (wrap in <label> outside instead)

* Things that know they are in a list (listIndex prop)

* Components that know about their parents' prop/state structure

#5

AVOIDING INSIDE-OUT COMPOSITION

#5: Avoiding inside-out composition

```
<div className="overlay" onClick={this._dismiss}>
    <div className="modal" onClick={this._ignore}>
        <UsernameInput /* ... */ />
        <PasswordInput /* ... */ />
        <div>
            <button onClick={this._submit}>
                Log In!
            </button>
        </div>
    </div>
</div>
```

#5: Avoiding inside-out composition

```
<div className="overlay" onClick={this._dismiss}>
    <div className="modal" onClick={this._ignore}>
        <UsernameInput /* ... */ />
        <PasswordInput /* ... */ />
        <div>
            <button onClick={this._submit}>
                Log In!
            </button>
        </div>
    </div>
</div>
```

#5: Avoiding inside-out composition

```
<Modal onClose={this._dismiss}>

        <UsernameInput /* ... */ />
        <PasswordInput /* ... */ />
        <div>
            <button onClick={this._submit}>
                Log In!
            </button>
        </div>

</Modal>
```

#5: Avoiding inside-out composition

Modal

```
<div className="overlay" onClick={this.props.onClose}>
    <div className="modal" onClick={this._ignore}>
        {this.props.children}
    </div>
</div>
```


FIN