Internet Applications Design and Implementation

2016 - 2017 - 2nd edition

(2 - Client applications React)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)

https://facebook.github.io/react/

React is component-based

- It allows to build encapsulated components that manage their own state
- Which can be composed to make complex UIs
- Since component logic is written in JavaScript instead of templates, one can easily pass rich data through the app and keep state out of the DOM

React is declarative and responsive

- Design simple views for each state in the app
- React efficiently updates and renders just the right components when data changes

The Key Ingredients

React components

- React components implement a **`render()`** method that **takes input data** and **returns what to display**

- The next example uses an XML-like syntax called JSX

- Input data that is passed into the component can be accessed by **`render()`** via **`this.props`**

- JSX is optional and not required to use React

```
var HelloMessage = React.createClass ({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
)}

ReactDOM.render(<HelloMessage name="Jane" />,
                document.getElementById("react_content"));
```

A react component with state

- A component can maintain internal state data

- Accessed via `this.state`

- When a component's state data changes, the rendered markup will be updated by re-invoking **`render()`**

```
var Timer = React.createClass ({
    getInitialState: function() {
        return {
            secondsElapsed: 0
        }
    },

    tick() {
        this.setState(function (prevState) {
            return {secondsElapsed: prevState.secondsElapsed + 1};
            }
        );
    },

    componentDidMount() {
        this.interval = setInterval(() => this.tick(), 1000);
    },

    componentWillUnmount() {
        clearInterval(this.interval);
    },

    render: function() {
        return (
                <div>
                    <HelloMessage name="Jane" />
                    Seconds elapsed since you arrived: {this.state.secondsElapsed}
                </div>
        )
    }
});

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

A complete reactive TODO app

TODO

- CIAI test 1
- CIAI test 2
- CIAI exam

CIAI 17/1

Add #4: CIAI 17/1

- Define how to render the component
- In this case a list (`<TodoList>`) and a form (`<form>`)

```
render: function() {
      return (
      <div>
         <h3>TODO</h3>
         <TodoList items={this.state.items} />
         <form onSubmit={this.handleSubmit}>
            <input onChange={this.handleChange} value={this.state.text} />
            <button>
                {'Add #' + (this.state.items.length + 1) + ': ' + this.state.text}
            </button>
         </form>
      </div>
      );
   },
```

- Define the initial state
- In this case a list of items a the current todo text

```
getInitialState: function() {
    return {
        items: [],
        text: ''};
  },
```

- Define what to do when the form is submitted
- In this case update the component's state by adding a tuple text item/date do the state list and resetting the current text

```
handleSubmit(e) {
    e.preventDefault();
    var newItem = {
        text: this.state.text,
        id: Date.now()
    };
    this.setState((prevState) => ({
      items: prevState.items.concat(newItem),
      text: ''
    }));
  }
```

- You can do more with the state to make your app reactive

- In this case, every time the item text changes, the button label also changes

- Can be used in any way as every time the state changes the component is rendered again

```
handleChange(e) {
      this.setState({text: e.target.value});
   },
```

```
var TodoApp = React.createClass ({
    getInitialState: function() {
        return {items: [], text: ''};
    },

    render: function() {
        return (
        <div>
            <h3>TODO</h3>
            <TodoList items={this.state.items} />
            <form onSubmit={this.handleSubmit} action=“”>
                <input onChange={this.handleChange} value={this.state.text} />
                <button>{'Add #' + (this.state.items.length + 1) + ': ' + this.state.text}</button>
            </form>
        </div>
        );
    },

    handleChange(e) {
        this.setState({text: e.target.value});
    },

    handleSubmit(e) {
        e.preventDefault();
        var newItem = {
                text: this.state.text,
                id: Date.now()
        };
        this.setState((prevState) => ({
            items: prevState.items.concat(newItem),
            text: ''
        }));
    }
});
```

- List the list element using a second component for more reusability

```
var TodoList = React.createClass({
  render: function() {
    return (
        <ul>
        {this.props.items.map(item => (
            <li key={item.id}>{item.text}</li>
        ))}
        </ul>
    );
  }
});

ReactDOM.render(<TodoApp />,
    document.getElementById("react_content"));
```

Ownership of components

- In React, an owner is the component that sets the **`props`** of other components
- More formally, if a component X is created in component Y's render() method, it is said that X is owned by Y
- Note that a component cannot mutate its **`props`**
- **`props`** are always consistent with what its owner sets them to
- This fundamental invariant leads to UIs that are guaranteed to be consistent
- In the todo example, TodoApp is the owner of TodoList

Ownership vs. Parenting

- There is a difference between the **owner-ownee** relationship and the **parent-child** relationship

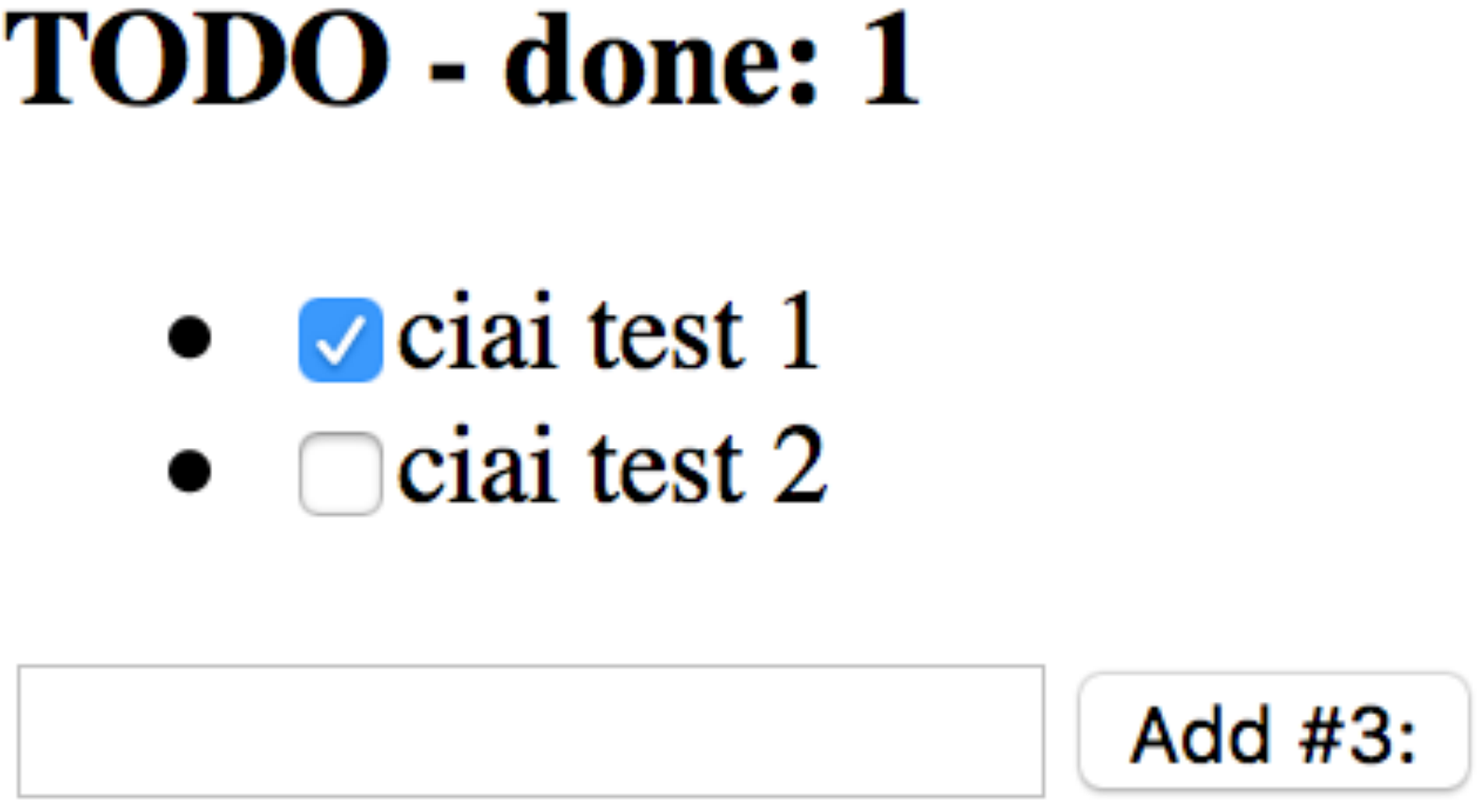
- The owner-ownee relationship is specific to React

- The parent-child relationship is simply the one you know from the DOM

- In the todo example TodoApp is the **owner** of h3, and TodoList and form is the **parent** of input

Inverse data flow or Two-way binding

- Components can only update their own state
- Ownees can only call owners update functions
- So how to do this?



```
var TodoList = React.createClass({
  handleItemClick: function(e) {
    if (e.target.checked)
      this.props.updateDone(1);
    else
      this.props.updateDone(-1);
  },

  render: function() {
    return (
        <ul>
        {this.props.items.map(item => (
          <li key={item.id}>
            <input type="checkbox" key={"done"+item.id}
              onClick={this.handleItemClick} />
            {item.text}
          </li>
        ))}
        </ul>
    );
  }
});
```

```
var TodoApp = React.createClass ({
   getInitialState: function() {
      return {items: [], text: '', done: 0};
   },

   render: function() {
      return (
         <div>
            <h3>TODO - done: {this.state.done}</h3>
            <TodoList items={this.state.items} updateDone={this.handleUpdateDone} />
            <form onSubmit={this.handleSubmit}>
               <input onChange={this.handleChange} value={this.state.text} />
               <button>{'Add #' + (this.state.items.length + 1) + ': ' +
                                    this.state.text}</button>
            </form>
         </div>
      );
   },
…
   handleUpdateDone: function (checked) {
      this.setState(
            function (prevState) {
               return {done: prevState.done + checked};
            }
      )
   }
});
```

Routing - setup

- No header incluir a biblioteca JS

```
<script src="https://unpkg.com/react-router/umd/ReactRouter.min.js"></script>
```

- Na aplicação React:

```
var Link = ReactRouter.Link;
var Router = ReactRouter.Router;
var Route = ReactRouter.Route;
```

https://css-tricks.com/learning-react-router/

Routing

```
var HelloMessageRouted = React.createClass ({
   render: function() {
      return (<div>Hello {this.props.params.name}</div>)
   }});

var SomeComponent = React.createClass ({
   render: function() {
      return (…
        <Link to={`/timer`}>Click to go to timer app</Link>
     …)
   }});

ReactDOM.render((
      <Router>
        <Route path="/" component={TodoApp}/>
        <Route path="/timer" component={Timer}/>
        <Route path="/hello/:name" component={HelloMessage}/>
       </Router>
      ), document.getElementById('react_content'))
```

React Thinking

https://facebook.github.io/react/docs/thinking-in-react.html

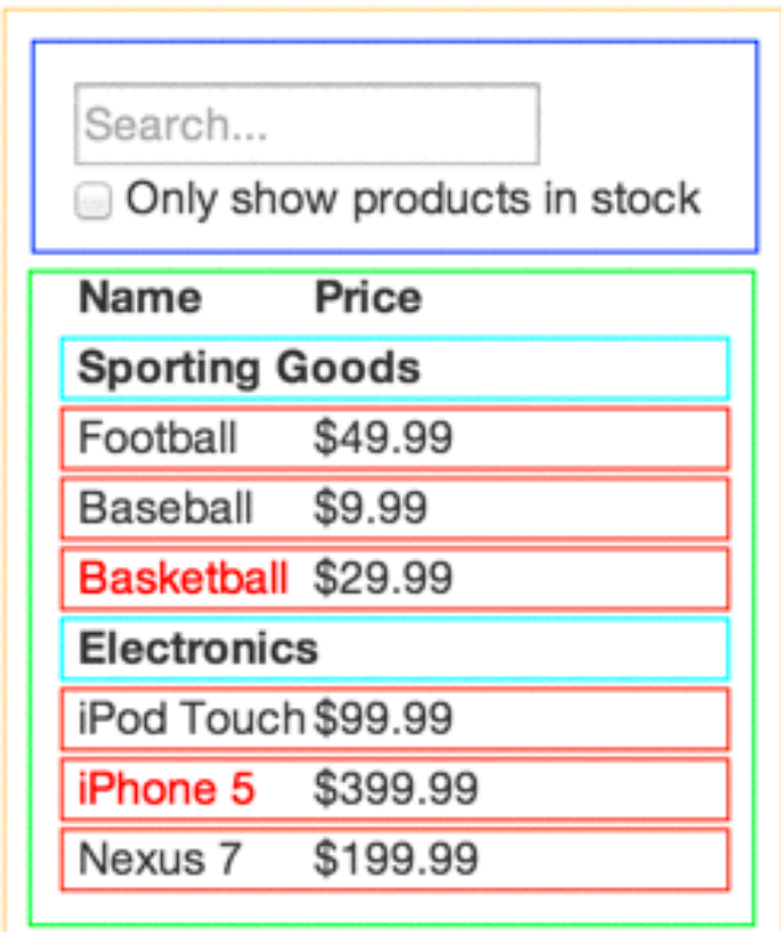
Break the UI into a component hierarchy

Identify the UI minimal complete state

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

State example

- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products



- Actual state:
  - The search text the user has entered
  - The value of the checkbox

Where should the state live?

- Probably not in the place you think…

- Form component? Think again…

- React is all about **one-way** data flow **down the component hierarchy**

- It may not be immediately clear which component should own what state

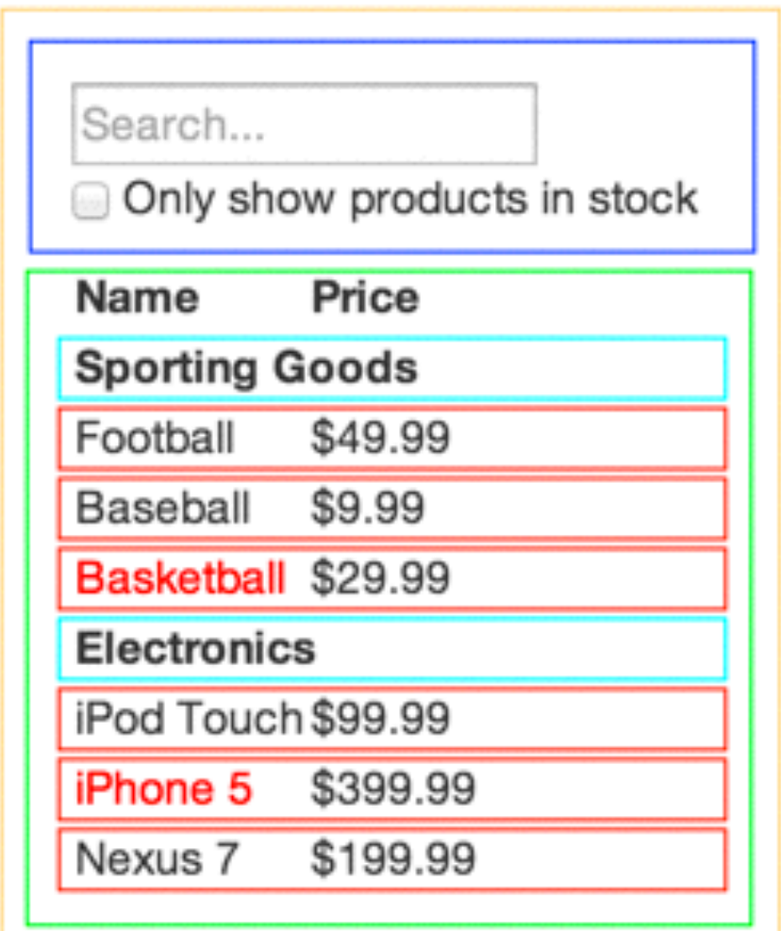
Find the state component by Q&A

For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

So, where should the state be?

- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products



- Actual state:
  - checkbox value: yellow component, complete app
  - search text: yellow component, complete app

Summary

- Create a component for each “part” of your app

- A component receives input (optional)

- And defines how it should be rendered (mandatory) in the **`render`** method

- Use **`this.props`** to access the inputs

Summary

- Optionally define a state to your component as a record (in `getInitialState`)
- Every time the component state changes the React runs the `render` method
- **Never** update the state directly!!
- Use `setState` to update it so Rect can properly propagate changes
- You can access the previous state just before updating it
- Just define as input for `setState` a function that receives the previous state and use it inside the function in any way you need

Summary

- Recall that React is all about one-way data flow

- Two make it two-way you need to explicitly state that in your code

- Owner components send a call-back function to ownees (through **`props`**) so they can update the owner state by calling the function