React

SWE 432, Fall 2018

Web Application Development

Review: What is state?

- All internal component data that, when changed, should trigger UI update
  - Stored as single JSON object this.state
- What isn't state?
  - Anything that could be computed from state (redundant)
  - Other components - should build them in render
  - Data duplicated from properties.

Review: Properties vs. State

- Properties should be **immutable**.
  - Created through attributes when component is instantiated.
  - Should *never* update within component
  - Parent may create a new instance of component with new prope

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

- State **changes** to reflect the current state of the component.
  - Can (and should) change based on the current internal data of your component.

Review: Handling events

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({ isToggleOn: !prevState.isToggleOn }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}

ReactDOM.render(
  <Toggle />, document.getElementById('root')
);
```

https://reactjs.org/docs/handling-events.html

Review: Component lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

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

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

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

Review: Component lifecycle

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(prevState => ({
      seconds: prevState.seconds + 1
    }));
  }

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

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

  render() {
    return (
      <div>
        Seconds: {this.state.seconds}
      </div>
    );
  }
}

ReactDOM.render(<Timer />, mountNode);
```

ReactDOM.render(...)
[component created]
constructor(...)
render()
componentDidMount()

tick()
render()
...

[component rendered again by parent]
componentWillUnmount()
[component created]
...

Review: Controlled Components

- Single source of truth
- Whenever a control changes its value
  - React is notified
  - State is updated
- Whenever state is updated
  - If necessary, render function executes and generates control with new value

Review: Reconciliation

- Process by which React updates the DOM with ea[illegible]ass
- O[illegible]based on order of co[illegible]ents
  - Second child of Card is destroyed.
  - First child of Card has text mutated.

```
<Card>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</Card>
```

```
<Card>
  <p>Paragraph 2</p>
</Card>
```

https://reactjs.org/docs/reconciliation.html

Today

- Reminders:
  - HW2 due tomorrow
  - Three more programming-related lectures, then move into usability
- Today:
  - Q&A on Async, React
  - React design patterns
  - In class React activity
  - Front end routing w/ react-router

Asynchronous Reminders

Questions about React

Handling events from collections of controls

```
class Board extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      squares: Array(9).fill(null),
    };
  }

  handleClick(i) {
    const squares = this.state.squares.slice()
    squares[i] = 'X';
    this.setState({squares: squares});
  }

  renderSquare(i) {
    return (
      <Square
        value={this.state.squares[i]}
        onClick={() => this.handleClick(i)}
      />
    );
  }
```

```
  render() {
      const status = 'Next player: X';

      return (
        <div>
          <div className="status">{status}</div>
          <div className="board-row">
            {this.renderSquare(0)}
            {this.renderSquare(1)}
            {this.renderSquare(2)}
          </div>
          <div className="board-row">
            {this.renderSquare(3)}
            {this.renderSquare(4)}
            {this.renderSquare(5)}
          </div>
          <div className="board-row">
            {this.renderSquare(6)}
            {this.renderSquare(7)}
            {this.renderSquare(8)}
          </div>
        </div>
      );
    }
}
```

fetch

- Where's the best place to put a fetch request?
- When the page loads, all of the components will immediately be asked to render.
  - Don't want to block the render from happenning....
- But want the component to update itself after the data gets back.
- Solution: componentDidMount
  - Make the fetch request from componentDidMount
  - After data comes back, update state with new data
  - This will trigger component to re-render

In Class Activity

- In groups of 2-3
- Starting with the finished todo app from last week:
  - Add a “delete item” button
  - Add an “edit” button that converts an item into a text field, then lets you save changes to it
- Starting code, finished code available on course syllabus

Todo in React

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { items: [], text: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  render() {
    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}
        </button>
      </form>
    </div>
  );
}
```

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

  handleSubmit(e) {
    e.preventDefault();
    if (!this.state.text.length) {
      return;
    }
    const newItem = {
      text: this.state.text,
      id: Date.now()
    };
    this.setState(prevState => ({
      items: prevState.items.concat(newItem),
      text: ''
    }));
  }
}

class TodoList extends Component {
  render() {
    return (
      <ul>
        {this.props.items.map(item => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
```

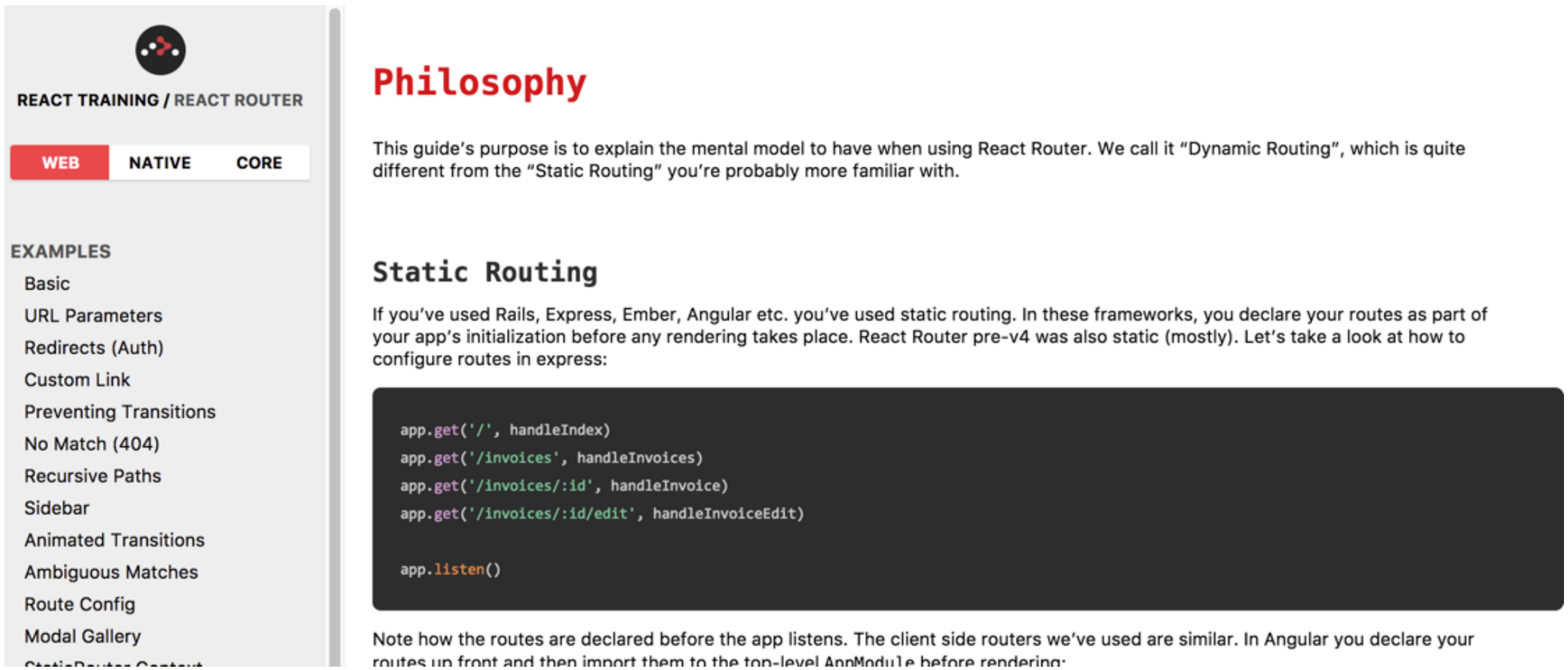
Front end routing

- Using state to represent views is great
- But....
  - Does not offer unique URL for each view
  - Breaks the back / forward buttons
  - Makes it harder to deep link to specific views

- Would be great to simply render a component based on the current URL
  - => front end routing

React-Router

- Install npm package in **client**
- `npm install react-router-dom`



https://reacttraining.com/react-router/web/guides/philosophy

In App

```jsx
import React from 'react'
import {
  BrowserRouter as Router,
  Route,
  Link
} from 'react-router-dom'

const Home = () => (
  <div>
    <h2>Home</h2>
  </div>
)

const About = () => (
  <div>
    <h2>About</h2>
  </div>
)

const Topic = ({ match }) => (
  <div>
    <h3>{match.params.topicId}</h3>
  </div>
)
```

```jsx
const Topics = ({ match }) => (
  <div>
    <h2>Topics</h2>
    <ul>
      <li>
        <Link to={`${match.url}/rendering`}>
          Rendering with React
        </Link>
      </li>
      <li>
        <Link to={`${match.url}/components`}>
          Components
        </Link>
      </li>
      <li>
        <Link to={`${match.url}/props-v-state`}>
          Props v. State
        </Link>
      </li>
    </ul>

    <Route path={`${match.url}/:topicId`} component={Topic}/>
    <Route exact path={match.url} render={() => (
      <h3>Please select a topic.</h3>
    )}/>
  </div>
)

const BasicExample = () => (
  <Router>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
        <li><Link to="/topics">Topics</Link></li>
      </ul>

      <hr/>

      <Route exact path="/" component={Home}/>
      <Route path="/about" component={About}/>
      <Route path="/topics" component={Topics}/>
    </div>
  </Router>
)
export default BasicExample
```