React Part 3

SWE 432, Fall 2019

Web Application Development

Review: Cascading selectors

- What happens if more than one rule applies?
- Most *specific* rule takes precedence
  - **p b** is more specific than **p**
  - **#maximizeButton** is more specific than **button**
- If otherwise the same, *last* rule wins
- Enables writing generic rules that apply to many elements that are overriden by specific rules applying to a few elements

Review: Centering content

```
.centered {
    width: 300px;
    margin: 10px auto 10px auto;
    border: 2px solid #0088dd;
}
```

This box is centered in its container.

- How do you center an element inside a container?
- Step 1: Must first ensure that element is *narrower* than container.
  - By default, element will expand to fill entire container.
  - So must usually explicitly set width for element.
- Step 2: Use *auto* value for left and right to create equal gaps

Review: Transitions

```
.box {
    width: 100px;
    height: 100px;
    background-color: #0000FF;
    transition: width 2s, height 2s, background-color 2s, transform 2s;
}

.box:hover {
    background-color: #FFCCCC;
    width: 200px;
    height: 200px;
    transform: rotate(180deg);
}

<div class="box"></div>
```

- transition: [property time], …, [property time]
  - When new class is applied, specifies the time it will take for each property to change
  - Can use *all* to select all changed properties

Review: Fixed width vs. liquid layouts

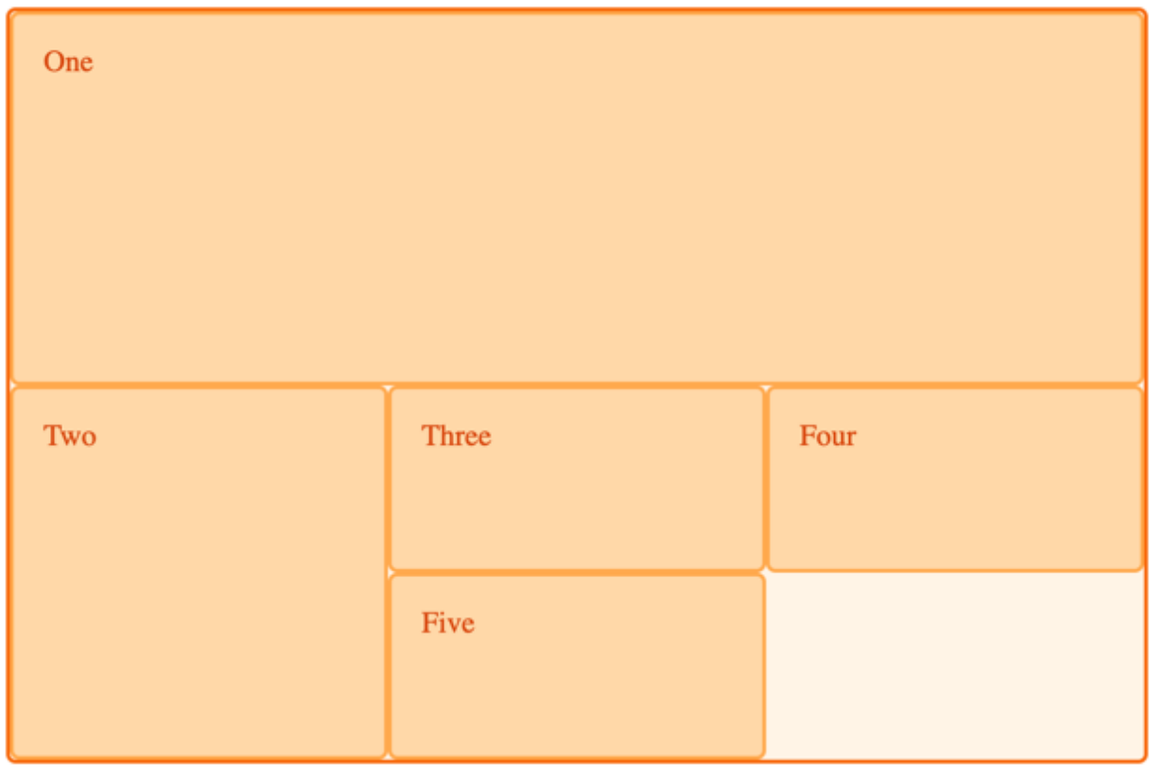
- Fixed width
  - Use width="[num]px" to force specific sizes
  - Allows for tightest control of look and feel
  - But can end up with extra whitespace around edge of web page
- Liquid layout
  - Use width="[num]%" to size relative to container sizes
  - Pages expand to fill the entire container size
  - Problems
    - Wide windows may create long lines of text can be difficult to read
    - Very narrow windows may squash words, breaking text onto many lines
  - (Partial) solution
    - Can use min-width, min-height, max-width, max-height to set bounds on sizes

Review: Display Grid

- Can explicitly place elements inside grid into grid areas

```
<div class="wrapper">
  <div class="box1">One</div>
  <div class="box2">Two</div>
  <div class="box3">Three</div>
  <div class="box4">Four</div>
  <div class="box5">Five</div>
</div>
```

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-auto-rows: 100px;
}
.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
.box2 {
  grid-column-start: 1;
  grid-row-start: 3;
  grid-row-end: 5;
}
```



https://jsfiddle.net/73dpfb4k/

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 each new render pass
- Occurs based on order of components
  - 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

Overview

- GUI Component Frameworks
- React Recap
- More React: React Router, working with state
- Demo

Logistics: HW4: Interactive Frontend

Requirements

- React
  - Create at least 5 separate React components.
  - Use conditional rendering to conditionally render visual content
  - Include handlers in your React components for at least 5 events
  - Create at least two controlled components, where input from an HTML control is bidirectionally synchronized with state in a React component
  - Create a list of child elements or components with unique keys
- CSS
  - Create at least one cascading selector which overrides another selector
  - Use at least two pseudo-classes
  - Center at least one element inside its container
  - Use the z-index and absolute or fixed positioning to display an element stacked on top of another element
  - Create an least one animation using transition
  - Specify at least one fixed size and one relative size
  - Use display grid to create a layout with multiple rows and columns
- **Due 11/18**

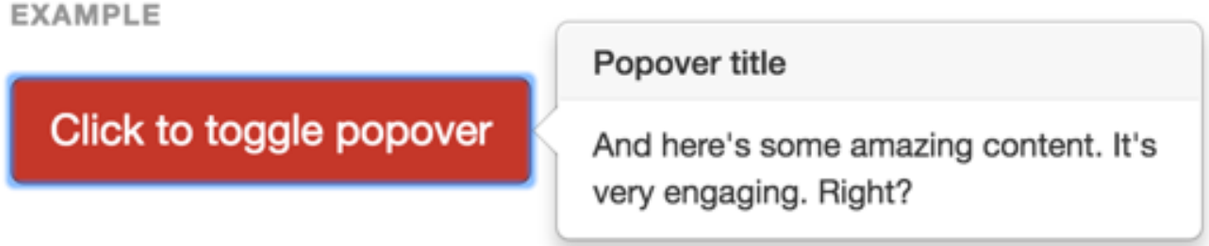
GUI Component Frameworks

- Can build arbitrarily complex UIs from the primitives we've seen
  - menus, nav bars, multiple views, movable panes, …
- But *lots* of work
  - Lots of functionality / behavior / styling to build from scratch
  - Browsers are not always consistent (*especially* before HTML5, CSS3)
  - Responsive layouts add complexity
- Solution: GUI component frameworks

GUI Component Frameworks

- Hig
  - F
  - E

EXAMPLE




```
<button type="button" class="btn btn-lg btn-danger" data-toggle="popover" title="Popover title"
data-content="And here's some amazing content. It's very engaging. Right?">Click to toggle
popover</button>
```

- Integrated component
  - Associate HTML elements with components using CSS classes
  - Framework dynamically updates HTML as necessary through JS
  - Offers higher-level abstractions for interacting with components

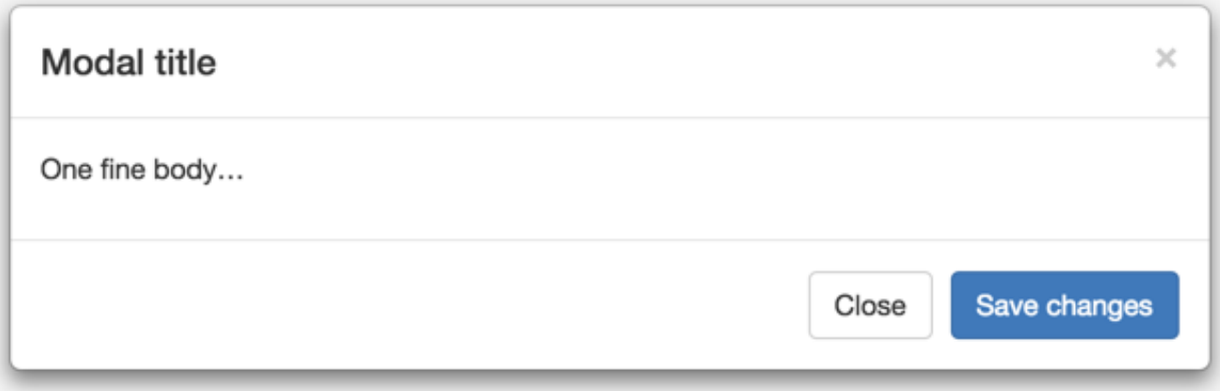
Bootstrap

- Popular GUI component framework
  - http://getbootstrap.com/
- Originally built and released by developers at Twitter in 2011
- Open source
- Offers baseline CSS styling & library of GUI components

Examples

Single toggle

```
<button type="button" class="btn btn-primary" data-toggle="button" aria-pressed="false"
autocomplete="off">
  Single toggle
</button>
```




```
<div class="modal fade" tabindex="-1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-
hidden="true">&times;</span></button>
        <h4 class="modal-title">Modal title</h4>
      </div>
      <div class="modal-body">
        <p>One fine body&hellip;</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div><!-- /.modal-content -->
  </div><!-- /.modal-dialog -->
</div><!-- /.modal -->
```

Bootstrap & React

- We'll use the react-bootstrap NPM module - Bootstrap for React!
- https://react-bootstrap.github.io

EXAMPLE

Holy guacamole! Best check yo self, you're not looking too good.

```
<Alert bsStyle="warning">
  <strong>Holy guacamole!</strong> Best check yo self, you're not looking too
  good.
</Alert>;
```

Questions about React

Select control example

https://codesandbox.io/s/dreamy-flower-3s536

More ways to create React components

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

equivalent to

```
const Square = props => {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

Conditional Rendering

- Based on state or props of component, render something

```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}
```

```
function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```

Lifting State Up

- If state is shared by multiple components (i.e., more than one component will change its rendering when the state changes), state should be shared in the least common ancestor component
- Key steps
  - Pass event handlers down: include param when creating leaf of event handler defined in component in ancestor
    - onSomethingChanged={this.changeHandler}
  - Listen for change: attach event handler for control change
    - onChange={this.handleChange}
  - Pass state change up: when event handler fires in leaf component, pass event up to parent
    - handleChange(e) {
    - this.props.onTemperatureChange(e.target.value);
    - }

Lifting state up: Example

https://codepen.io/gaearon/pen/WZpxpz?editors=0010

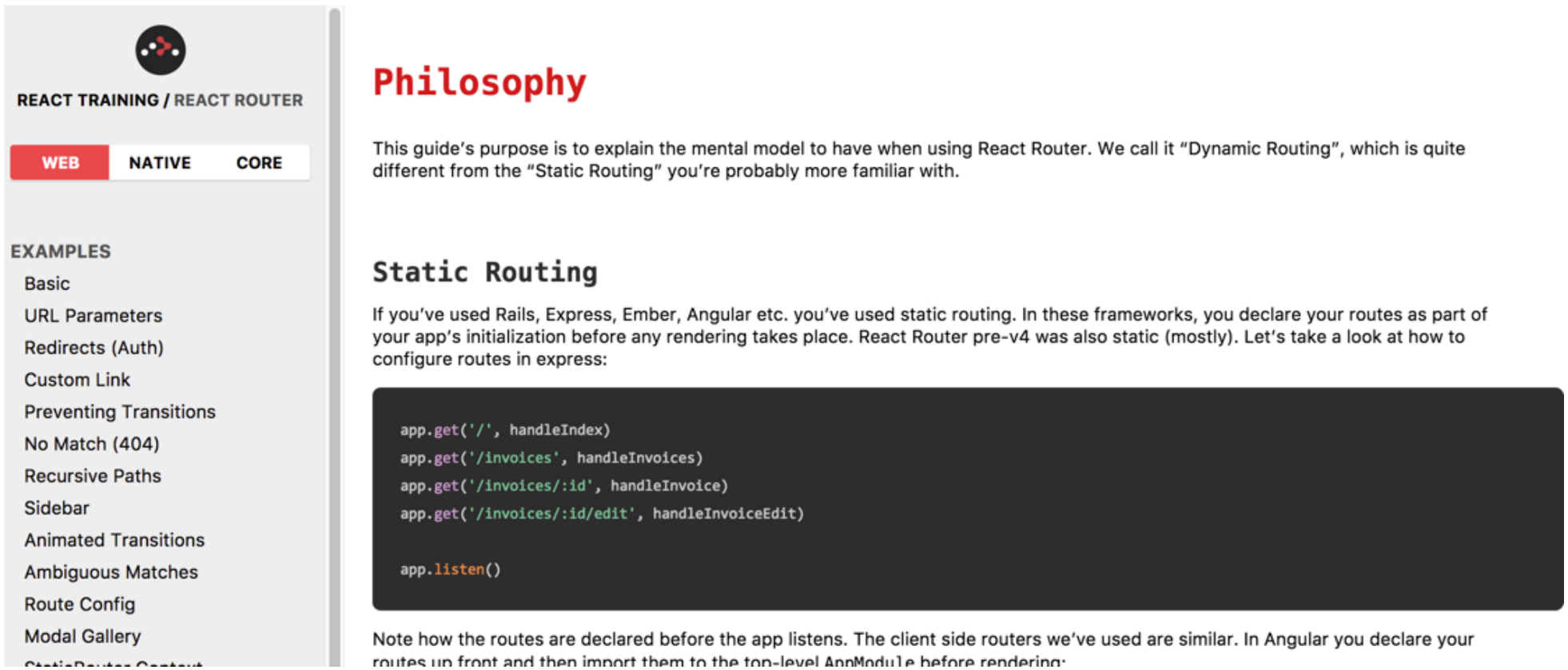
Front end routing

- 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

```
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>
)
```

```
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
```

Router example

- https://codesandbox.io/s/react-router-basic-bnpsd?from-embed