

SWE 432 -Web Application Development

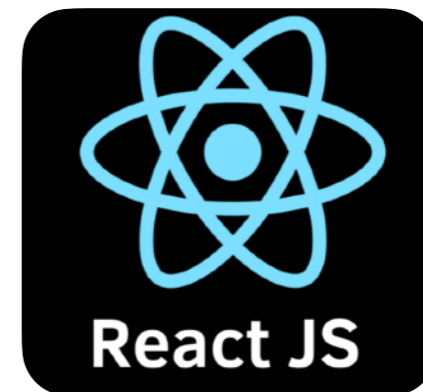
Fall 2022



George Mason
University

Dr. Kevin Moran

Week 10:
More React &
CSS





Administrivia

- *HW Assignment 3* - Due today, grades and comments will be posted by Thursday next week.
- *HW Assignment 4* - Out today, Due in three weeks (November 17th)
 - Extra Credit Opportunity!
- *Mid-Semester Course Feedback Survey*: Thank you to those who filled out the survey!



HW Assignment 4

Step 1: Sign up on GitHub Classroom to Clone the Starter Project

Please follow the instructions for setting up this homework assignment in GitHub Classroom and deployment of your project via Heroku. The starter project includes code for a React Front-End. You may reuse your code from HW2 or HW3 if you would like to use a backend for this assignment, however, this is not required.

[Click Here to View HW 4 Tutorial \(Coming Soon!\)](#)

Step 2: Choose an Idea for an Interactive App

In this assignment, you are free to choose any idea for an app you'd like as long as it involves user interactivity, where the application is taking input from the user (e.g., clicking on buttons, entering text), updating the state of the application, and rendering new visual content from the new state. You might create a simple game, such as a number guessing game or checkers. You might create a data management app to, for example, create and browse recipes or track expenses. You should pick something that is interesting and exciting to you.



HW Assignment 4

Step 3: Implement your Interactive App

You'll implement your app as a front-end React app. Your app should satisfy the following requirements:

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 at 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
 - It is optional to use any styling libraries like Bootstrap or Material-UI, however, you must manipulate CSS as required above (for example, customizing the library with your own CSS files, or inline by setting style within your React components).



HW Assignment 4 - Extra Credit

I will be offering Extra Credit on this assignment. This extra credit will be applied toward your midterm exam grade, with a maximum of 10 potential points to be earned (out of 200). This is essentially amounts to half a letter grade.

- **Bonus (Extra Credit)**
 - Create at least 5 jest tests to test the functionality of your app
 - You should use at least 3 tests that use the `jest-dom` package and at least 2 "normal" Jest tests



Class Overview

- **Part 1:** CSS & DOM
 - Finish Lecture from Last Class
- **Part 2:** More React Techniques!
 - Quick Lecture
 - Hands-On Session

Finishing Up CSS Concepts

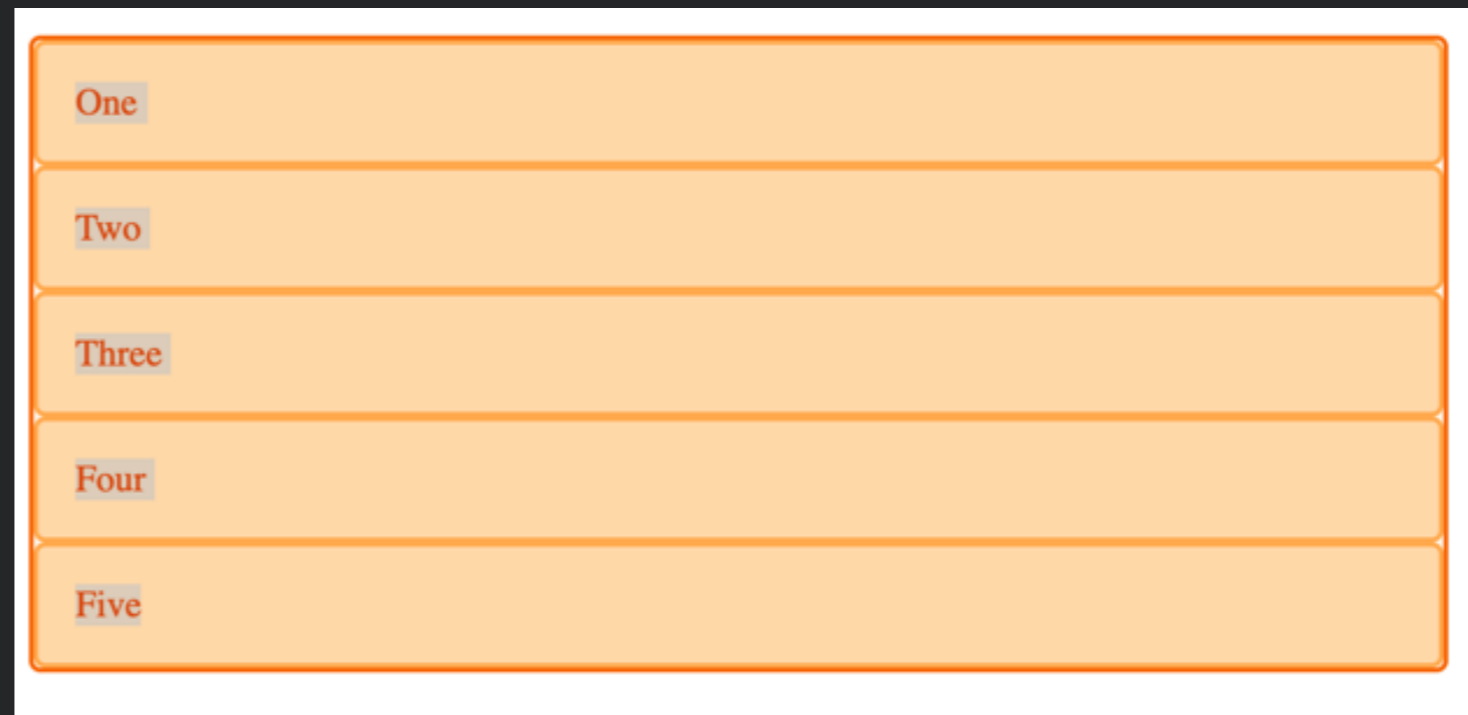


Grid layout

- Create using display: grid or display: inline-grid

```
<div class="wrapper">  
  <div>One</div>  
  <div>Two</div>  
  <div>Three</div>  
  <div>Four</div>  
  <div>Five</div>  
</div>
```

```
.wrapper {  
  display: grid;  
}
```



Grid tracks

- Define rows and columns on grid with the *grid-template-columns* and *grid-template-rows* properties.
- Define grid tracks.
- A grid track is the space between any two lines on the grid.

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```



Liquid layouts

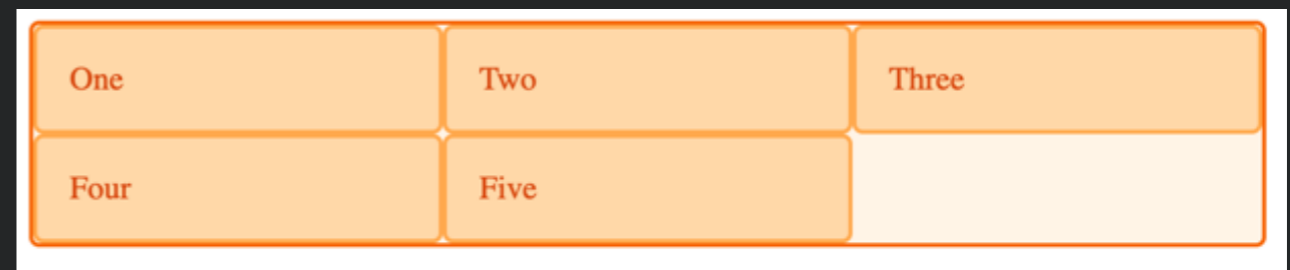
- `fr` represents a fraction of available space for grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```

<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}

```



```

.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}

```

Liquid layouts

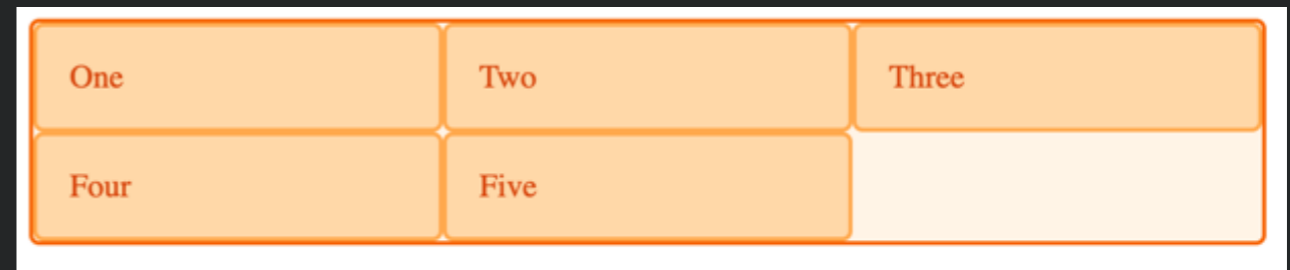
- fr represents a fraction of available space for grid container.
- Can mix absolute and flexible, where flexible occupies any remaining space after flexible is subtracted

```

<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

```



```

.wrapper {
  display: grid;
  grid-template-columns: 500px 1fr 2fr;
}

```

Positioning items

- 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;
}
```



- Can set gaps between columns and rows

```
<div class="wrapper">
  <div>One</div>
  <div>Two</div>
  <div>Three</div>
  <div>Four</div>
  <div>Five</div>
</div>

.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  column-gap: 10px;
  row-gap: 1em;
}
```

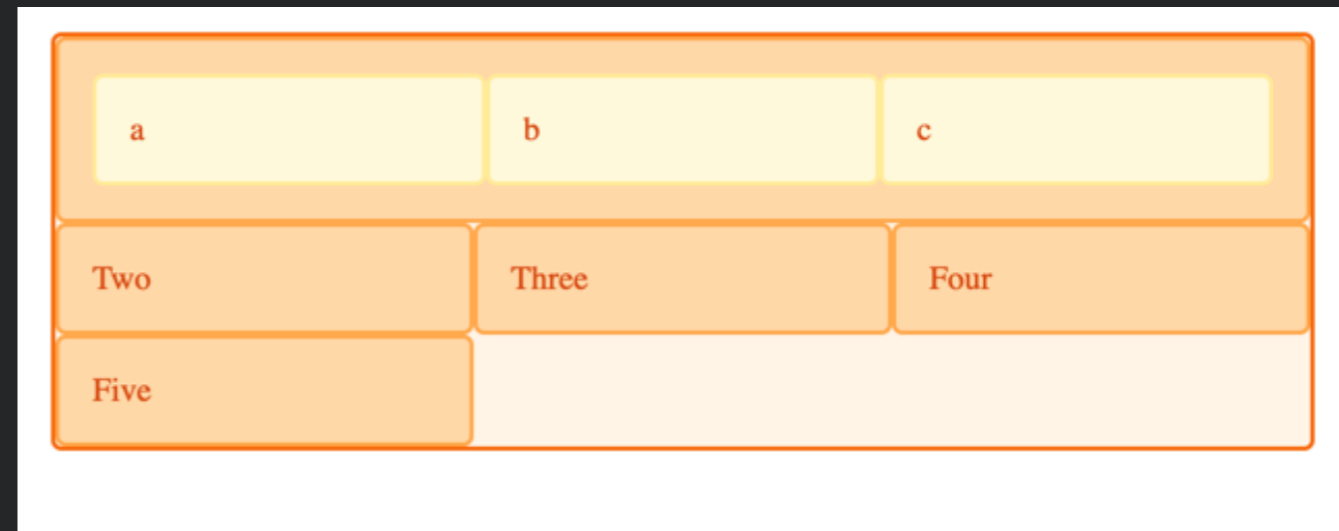


Nesting

- Can nest grids, which behave just like top-level

```
<div class="wrapper">
  <div class="box box1">
    <div class="nested">a</div>
    <div class="nested">b</div>
    <div class="nested">c</div>
  </div>
  <div class="box box2">Two</div>
  <div class="box box3">Three</div>
  <div class="box box4">Four</div>
  <div class="box box5">Five</div>
</div>

.box1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
}
```





Designing for mobile devices

- Different devices have different aspect ratios.
 - Important to test for different device sizes.
 - May sometimes build alternative layouts for different device sizes.
- Using specialized controls important.
 - Enables mobile browsers to use custom device-specific widgets that may be much easier to use.

<small>Mon 4 Nov</small>	<small>12</small>	<small>56</small>
Tue 5 Nov	13	57
Wed 6 Nov	14	58
Thu 7 Nov	15	59
Today	16	00
Sat 9 Nov	17	01
Sun 10 Nov	18	02
Mon 11 Nov	19	03
<small>Tue 12 Nov</small>	<small>20</small>	<small>04</small>



CSS Best Practices

- When possible, use CSS to declaratively describe behavior rather than code
 - Easier to read, can be optimized more effectively by browser
- Don't repeat yourself (DRY)
 - Rather than duplicating rules, create selectors to style all related elements with single rule
- CSS should be readable
 - Use organization, indentation, meaningful identifiers, etc.

Review of Previous React Concepts





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')
);
```



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

```
class EssayForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      value: 'Please write an essay about your favorite DOM element.'
    };

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('An essay was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```



Functional Components + Hooks

But what if we want state + clean functional components??

```
import React, { Component } from 'react';

class Counter extends Component {
  constructor(props) {
    super(props);

    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
          onClick={() =>
            this.setState({ count: this.state.count + 1 })
          }
        >
          Click me
        </button>
      </div>
    );
  }
}

export default Counter;
```



Functional Components + Hooks

Now we can have both with functional components + hooks!

```
import React from 'react';

// how to use the state hook in a React function component
function Counter() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

export default Counter;
```



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

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

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

- 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.

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

More React Programming



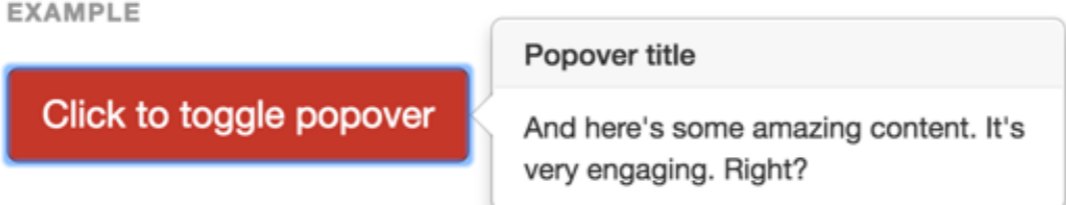


GUI Component Frameworks

- 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

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>
```

- Higher-level abstractions for GUI components
 - Rather than building a nav
 - Exposes new options, events, properties
- 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



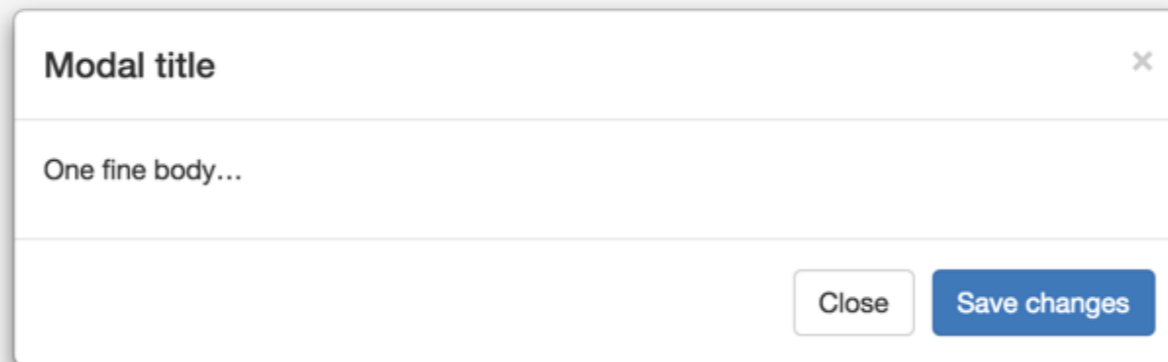
Bootstrap

- 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>;
```



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>;
```



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 />;
}
```

Example: <https://replit.com/@kmoran/swe-432-cond-render>



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

```
npm install react-router-dom
```

REACT TRAINING / REACT ROUTER

WEB NATIVE CORE

EXAMPLES

- Basic
- URL Parameters
- Redirects (Auth)
- Custom Link
- Preventing Transitions
- No Match (404)
- Recursive Paths
- Sidebar
- Animated Transitions
- Ambiguous Matches
- Route Config
- Modal Gallery

Philosophy

This guide's purpose is to explain the mental model to have when using React Router. We call it "Dynamic Routing", which is quite different from the "Static Routing" you're probably more familiar with.

Static Routing

If you've used Rails, Express, Ember, Angular etc. you've used static routing. In these frameworks, you declare your routes as part of your app's initialization before any rendering takes place. React Router pre-v4 was also static (mostly). Let's take a look at how to configure routes in express:

```
app.get('/', handleIndex)
app.get('/invoices', handleInvoices)
app.get('/invoices/:id', handleInvoice)
app.get('/invoices/:id/edit', handleInvoiceEdit)

app.listen()
```

Note how the routes are declared before the app listens. The client side routers we've used are similar. In Angular you declare your routes up front and then import them to the top-level AppModule before rendering:

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



Router example

- <https://codesandbox.io/s/react-router-basic-forked-fite7q?file=/example.js>

Functional React & CSS Exercises

- React Exercise:
 - Instructions: Implement conditional rendering so that the Message is displayed when a user presses the button.
- CSS Exercise:
 - Instructions:
 - Center a component inside it's container
 - Use a display grid to create layout with multiple rows and columns
 - Override one of the Bootstrap selectors
 - <https://replit.com/@kmoran/swe-432-react-example#src/App.jsx>

Please Turn-in Your In-Class Activity for a Quiz Grade



<https://bit.ly/3SDGckh>

Please provide the URL of your Replit Repo



Acknowledgements

- Slides Adapted from Dr. Thomas LaToza's SWE-432 course