Lectures 11 and 12:

React for JavaScript

05-431/631 Software Structures for User Interfaces (SSUI)

Fall, 2022

Logistics (lecture 11)

- No longer need to create GitHub Pages version for homeworks.
- No office hours tomorrow (Wednesday, Oct. 5)
- Starting to cover React today, since HW4 spans mid-fall break, when there are no lectures or labs

- Midterm quiz
  - Take home, open book, open internet
  - ~~Will start at 3:05 on Wednesday 10/12/2022, due 24 hours later on Thursday, 10/13/2022 at 3:05 (before class)~~
    - ***Updated: Wednesday 10/12/2022, due 48 hours later on Friday, 10/14/2022 at 3:05 but still come to class on Thursday***
  - Will include topics through lecture 12 (formerly 13), and homeworks 1 to 3

Logistics (lecture 12)

- These slides were updated
  - Reload slides from schedule, lecture 11
- HW3 due Tuesday
- Bug in supplied floodfill js file
  - See Piazza
  - Or download new version: https://github.com/CMU-SSUI-Fall2022/HW3/blob/main/floodfill.js
- Request for midterm to be Thursday after class until Friday:
  - 4:25 (after class) on Thursday, 10/13/2022, due 24 hours later, on Friday, 10/14/2022 at 4:25
  - *Any objections?*

What is React

- https://reactjs.org/
- "A JavaScript library for building user interfaces"
- Created by Facebook and actively used and supported by them
- Goal: be more declarative, like original html and CSS
  - As opposed to imperative like JavaScript
  - React handles updating and redrawing as data changes
    - Still need input handlers
  - New way to write html with computed parameters = JSX
- Build reusable, encapsulated components
  - E.g., header and footer
- React can also be used "native" to make Android and iOS apps

Key Concepts

- JSX – yet another syntax for html
  - Almost like regular html, but some differences
    - Yet another syntax for comments: `{\* comment *\}`
  - Compute elements based on `props`
  - Computed and returned by `render()` methods
  - CSS classes called `className`
- States – dynamic data
  - Note: totally different from state-transition-diagrams
  - Store values as JS objects
  - DOM updated when state's values change

LIVE JSX EDITOR

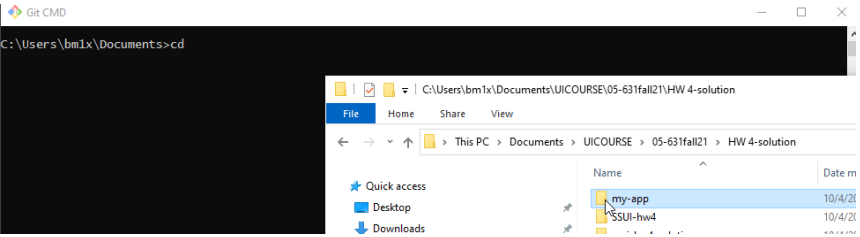
```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

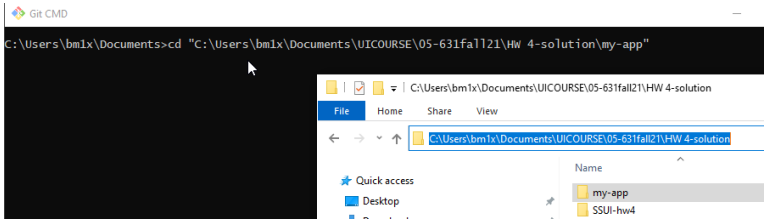
Getting Started

- Download node.js and npm: https://nodejs.org/en/
- Now can bring up a console and type `npm` or `npx`
- Follow these instructions for getting started with React
  - https://create-react-app.dev/docs/getting-started#quick-start
- cd into the code folder *(expert hint: use drag-and-drop – see below)*

```
C:\Users\bm1x\Documents> cd C:\Users\bm1x\Documents\UICOURSE\05-631fall22\HW 4-solution
C:\Users\bm1x\Documents\UICOURSE\05-631fall22\HW 4-solution> npx create-react-app my-app
C:\Users\bm1x\Documents\UICOURSE\05-631fall22\HW 4-solution> cd my-app
C:\Users\bm1x\Documents\UICOURSE\05-631fall22\HW 4-solution> npm start
```

  - Will open new tab in browser running the app.
  - Can edit App.js and changes will be shown immediately on save
  - Stop with ^C in cmd window, or close tab, or close cmd window





Extra resources for getting started

- https://reactjs.org/docs/create-a-new-react-app.html
- https://docs.npmjs.com/downloading-and-installing-node-js-and-npm
- W3Schools intro on how to get started with React dev: https://www.w3schools.com/react/react_getstarted.asp

- Thanks to Michael Liu and Clara Cook and Alex Cabrera also for help with these slides!

Learning React

- Can't cover it all in two lectures & two labs
- Lots of great material out there to learn from
- Lots of Stack Overflow answers that cover most problems
- eCommerce websites are exactly its target application
- HW4 write up includes an example app from Michael Liu that is similar to HW4: SSUI-Star-Wars
- How to deploy React+Router app to Netlify from GitHub: https://dev.to/easybuoy/deploying-react-app-from-github-to-netlify-3a9j

JavaScript features heavily used by React

- *Might want to review these features!*
- export and import
  - React uses lots of files and need to control namespaces
- spread and rest operator: . . .
  - Flatten an array or object in place

```
const oldArray = [4,5];
const newArray = [...oldArray, 1, 2, 3];
      newArray = [4,5,1,2,3] instead of [ [4,5], 1,2,3]

const newObject = {...oldObject, newProp: 'Jason'}
function sortNumbers(...args) { }  //args is an array of the parameters
```

- Destructuring – assign variables using same name

```
let {name, loc} = func(); // returns object with name: and loc: fields
```

Useful JavaScript array functions

- `ar1.map(fn)` – returns a new array of calling fn on each element of ar1

```
const doubleNumArray = numbers.map(num => num * 2);
```

- `ar2.filter(fn)` – returns a new array containing the elements of ar2 that when passed to fn return true

```
const longwords = words.filter(word => word.length > 6);
```

Backquote ("backtick") operator

- *(Covered in Lab)*
- Called "template literals", ref: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
- Creates a string, where elements in ${} are evaluated
  - `document.title = `You clicked ${count} times`;`
  - Same as: `"You clicked " + count + " times";`
- Can also include newlines:

```
`string text line 1
 string text line 2`
```

is the same as:

```
'string text line 1\n' +
'string text line 2'
```

React is Client Side

- Compiled into pure JavaScript
- But uses lots of libraries
- npm to manage everything for you
  - https://www.npmjs.com/
  - Originally: **Node Package Manager**
  - Command line interface
  - Requires using a server on your machine
    - Automatically recompiles when edit code
- Start with React `create-react-app`

JSX

- React's version of html
- Describe the desired code in-line, without needing strings
- Compute parts based on variables, lists, etc. by putting it in {}
- Note, *not* strings
- Usually in a `return()` or an assignment

```
return(
   <div>
     <h1>Hello, world!</h1>
     <h2>It is {new Date().toLocaleTimeString()}.</h2>
   </div>
);
const element = (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is {new Date().toLocaleTimeString()}.</h2>
        </div>
);
```

React Components

- Elements of the web page
- Styled with regular CSS
- Defined using jsx
- Use “className” for CSS classes

```
<div id="p1"></div>
<div id="p2"></div>
<div id="p3"></div>
```

```
.person {
 margin: 10px;
 border: 1px solid #aaa;
 background-color: lightyellow;
 padding: 5px 10px;
 width: 250px;
 box-shadow: 0 4px 4px #888;
 margin: 30px 20px;
}
```

```
const Person = (props) => {
 const {name, location, birthday} = props
  return (
   <div className="person">
     <h1>{name}</h1>
     <h3>Location: {location}</h3>
     <h3>Birthday: {birthday}</h3>
   </div>
 )
}

ReactDOM.render(<Person name={"Jason"}
location={"Pittsburgh"} birthday={"August 4"}/>,
document.querySelector('#p1'))

ReactDOM.render(<Person name={"Michael"}
location={"Pittsburgh"} birthday={"June 12"}/>,
document.querySelector('#p2'))

ReactDOM.render(<Person name={"Elon"} location={"LA"}
birthday={"June 28"}/>,
document.querySelector('#p3'))
```

Jason

Location: Pittsburgh

Birthday: August 4

Michael

Location: Pittsburgh

Birthday: June 12

Elon

Location: LA

Birthday: June 28

Structure of our React apps

- index.html just contains `<div id="root"></div>`
- index.js connects the app component to it:

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

- Always need a “render” method at each level
  - Takes function that returns contents (App) and where to put it (‘root’)
- App.js – contains the content of the app by importing all the components

Create components

- React components: custom HTML elements that can be used to construct a web app
  - Create your elements
- One React app typically only has ONE root component. In our case, it's the `App` component.
- Nest all the other components in the root (App) component => a tree of components
  - Can use `<div>` like usual

```
<div className="App">
      xxx
      <img src xxx />
      <p> xxx </p>
</div>
```

- Each component needs to return/render some JSX code - it defines which HTML code React should render to the DOM in the end
  - Instead of constructor, React classes have a `render()` method

Dynamic Contents: Props and State

- Props – variable that holds attributes to be used
- Passed down hierarchy – parent to child

```
  render() {
    return (
      <div className="App">
        <h1>Hi, I'm a React app</h1>
        <Person name="Michael" location="Pittsburgh" />
        <Person name="Elon" location="LA" />
        <Person name="Jason" location="Pittsburgh">
          He just had a new baby!
        </Person>
      </div>
    );
  }
}
```

```
import React from 'react';

const person = props => {
  return (
    <div>
      <h3>
        I'm {props.name}! I'm in {props.location}
      </h3>
      <p>{props.children}</p>
    </div>
  );
};

export default person;
```

Dynamic Contents: Props and State

- State: Variables that change, causing redisplay
  - Note: *not* MVC model with listener pattern
  - Shared variables instead
- Both props and state are JavaScript objects
  - attribute-value pairs

```
  state = {
    persons: [
      { name: 'Michael', location: 'Pittsburgh' },
      { name: 'Elon', location: 'LA' },
      { name: 'Jason', location: 'Pittsburgh' },
    ],
  };

  render() {
    return (
      <div className="App">
        <h1>Hi, I'm a React app</h1>
        <Person name={this.state.persons[0].name} location={this.state.persons[0].location} />
        <Person name={this.state.persons[0].name} location={this.state.persons[0].location} />
        <Person name={this.state.persons[0].name} location={this.state.persons[0].location}>
          He just had a new baby!
        </Person>
      </div>
    );
  }
}
```

this.setState or useState

- Use `this.setState` to change state if using classes
  - Cannot update state directly
  - Challenging when state is data structure
- Or `const [count, setCount] = useState(0);` if using hooks (see below)
- So React knows to calculate what to redisplay
- Minimize the number of states
- Calculate props (attributes) based on state

Conditional rendering

- Can use ternary to compute contents, typically based on state
  - null causes nothing to be displayed
- Can just compute contents
  - True means it will get rendered
  - Same as ternary but…
    - && instead of ?
    - Null need not be specified

```
state = {
  persons: [
    { name: 'Michael', location: 'Pittsburgh' },
    { name: 'Elon', location: 'LA' },
    { name: 'Jason', location: 'Pittsburgh' },
  ],
  showPersons: false,
};
```

```
togglePersonsHandler = () => {
  let showPersons = !this.state.showPersons;
  this.setState({ showPersons: showPersons });
};
```

```
<button style={buttonStyle} onClick={() => this.togglePersonsHandler()}>
  Toggle Persons
</button>

{this.state.showPersons ?
  <div>
    <Person name={this.state.persons[0].name} location={this.state.persons[0].location} />
    <Person
      name={this.state.persons[1].name}
      location={this.state.persons[1].location}
      click={this.switchNameHandler}
      changed={this.nameChangeHandler}
    />
    <Person name={this.state.persons[2].name} location={this.state.persons[2].location} />
  </div>
) : null}
```

Rendering Lists

- Use `map()` to return jsx from data list
- Don't forget the "key" attribute -- needed for efficient redisplay
  - Must be unique within the list
  - Should be "stable" wrt the element
  - React uses it to compute what has changed

```
state = {
  persons: [
    { name: 'Michael', location: 'Pittsburgh' },
    { name: 'Elon', location: 'LA' },
    { name: 'Jason', location: 'Pittsburgh' },
  ],
  showPersons: false,
};
```

```
<div>
  {this.state.persons.map((person, idx) => {
    return <Person key={idx} name={person.name} location={person.location} />;
  })}
</div>
```

Two ways to use React

- Two ways to use React
  - Tutorials and other materials online do not distinguish them
    - Confusing mix of approaches
  - No clear names for them
- Object-based or function-based
  - Function-based is newer – relies on “hooks”
  - Both are considered acceptable in general
  - Alex will cover these more tomorrow as well

Object Based

- Create a new class that provides a render method (instead of a constructor)
- Pass this class as a parameter to ReactDOM.render()
  - Also pass the place in the DOM to display it.

LIVE JSX EDITOR

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

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Function Based

- Create a new function instead of a new class
- Make use of “hooks” to connect into DOM

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

React Hooks

- https://reactjs.org/docs/hooks-intro.html
- Added in React 16.8 released February 2019
- Replaces need to define new classes
- "Hooks are functions that let you "hook into" React state and lifecycle features from function components."
- Simpler way to assign and access state variables
  - State hook and Effects Hook
- Historical note: "hooks" date at least back to the 1970s
  - E.g., Emacs hooks – functions you can assign that will be run just before or after some important event happens

State Hook

- `useState` is the state hook
- Takes initial value of state
- Returns: current value of state, and function to update it
- `count`, `setCount` can be any pair of names
- Values are remembered across executions

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

State Hook, cont.

- Must be defined in top-level functions
- Initial value can be a single value or object
- If multiple `useState()` they must be *in the same order* everywhere
  - Can use any names
- Be sure to put uses in `{ }` so evaluated

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

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

Effect Hook

- Takes a function that will be called at a specific time
  - By default, *after* every render (including the first time)
- Can perform side effects – updating external things
- Can use state and other variables since inside the top-level function.
- The function parameter can return another function to be used as cleanup
  - When the component "unmounts" (goes to another page)

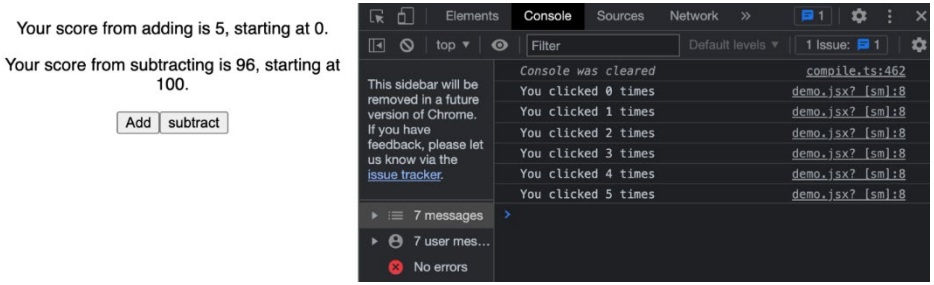
```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

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

```
useEffect(() => {
  ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
  return () => {
    ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
  };
});
```

Effect Hook (another example)

- `useEffect()` can take dependencies
  - Dependencies placed at in brackets at the end of function
  - Dependencies say that only when the specified variables change, will the app remount
- Augment our current example, so that we have 2 state variables:
  - Only when add is updated, does it print to console

```
import React, { useState, useEffect } from "react";

export default function Example() {
  const [add, setAdd] = useState(0);
  const [subtract, setSubtract] = useState(100);

  useEffect(() => {
    console.log(`You clicked ${add} times`);
  }, [add]);

  return (
    <div>
      <p>Your score from adding is {add}, starting at 0.</p>
      <p>Your score from subtracting is {subtract}, starting at 100. </p>
      <button onClick={() => setAdd(add + 1)}> Add</button>
      <button onClick={() => setSubtract(subtract - 1)}> subtract</button>
    </div>
  );
}
```



Props parameter

- Props can be passed in 2 ways
  - Still will be passed in same way as class components [picture A]
  - Direct use of props (notice you no longer need “this” anymore) [picture B]
    - ”this” is considered inherent
  - Object that contains parameters passed as props [picture C]
    - Separated by commas

A

```
import "./styles.css";
import Example from "./demo";

export default function App() {
  return (
    <div className="App">
      <Example name="apollo" tvShow="Battlestar Galactica" />
    </div>
  );
}
```

B

```
export default function Example(props) {
  return (
    <div>
      <p>This page was made by {props.name}</p>
      <p>They love watching {props.tvShow}</p>
    </div>
  );
}
```

C

```
export default function Example({ name, tvShow }) {
  return (
    <div>
      <p>This page was made by {name}</p>
      <p>They love watching {tvShow}</p>
    </div>
  );
}
```

Passing function as props

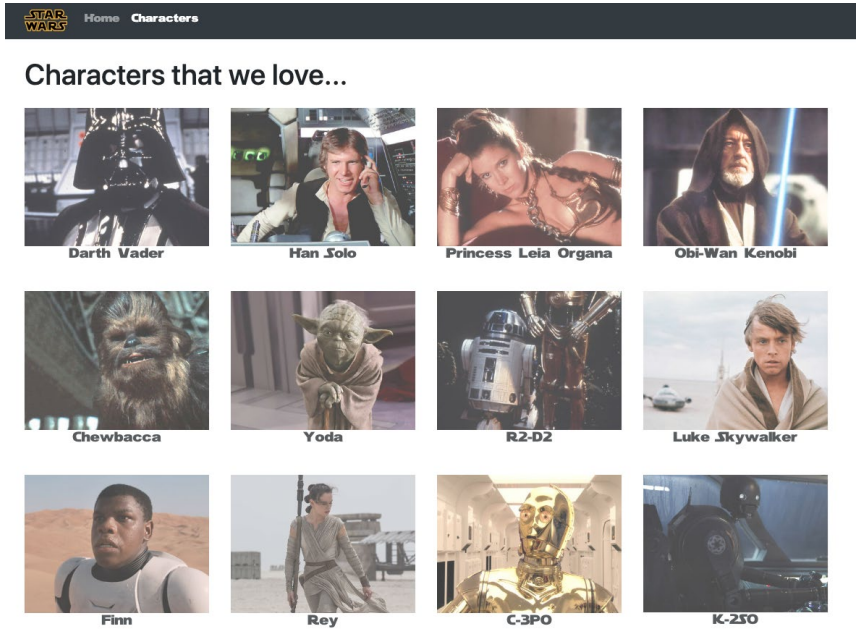
- This is the same example as the class component
- Constructor not necessary
  - Automatic Binding (why “this” is not necessary either)
  - Child component does not need “this” either

```
import "./styles.css";
import Example from "./demo";

export default function App() {
  let handleClick = (id) => {
    console.log(id);
  };

  return (
    <div className="App">
      <Example onclick={handleClick} />
    </div>
  );
}
```

```
import React, { useState, useEffect } from "react";

export default function Example(props) {
  const [thingToSay, setThingToSay] = useState("woo");

  return (
    <div>
      <button onClick={() => props.onclick(thingToSay)}>Log Woo!</button>
    </div>
  );
}
```

Michael Liu's sample app

- https://lxieyang.github.io/ssui-simple-react-app
  - Updated to React v18 – nothing important is new (yet!) -ref
- Basic code structure
- Component Reuse
- Lists
- Uses Router
- Styling using ReactStrap
- *But doesn't use states*



Extra Packages You Can Use

- React Router
  - Provides for multiple pages with unique URLs
  - Can pass parameters from one page to another so don't need localStorage
    - Useful for which shirt clicked on
    - https://reactrouter.com/en/main
- ReactStrap
  - Helpful formatting and layout mechanisms so you don't need to make them from scratch
  - Based on Bootstrap but updated for React
  - https://reactstrap.github.io/

- *(But no others for homework 4!)*

React Router

- https://reactrouter.com/en/main
- Support multiple pages with same header/footer
  - react-router-dom – allows switch among different content

```
const app = () => {
  return (
    <div className="App">
      <NavBar />

      <div className="MainContent">
        <Switch>
          <Route exact path={appRoutes.home}>
            <HomePage />
          </Route>
          <Route exact path={appRoutes.characters}>
            <CharactersPage />
          </Route>
          <Route exact path={appRoutes.character}>
            <CharacterPage />
          </Route>
          <Redirect to={appRoutes.home} />
        </Switch>
      </div>

      <Footer />
    </div>
  );
};
export default app;
```

ReactStrap

- https://reactstrap.github.io/
- Containers for layout
  - <Container> … </Container>
  - Row
  - Col
- Simple parameterizations to make Responsive to window sizes:

```
<Col lg={4} md={6} sm={12}>
```

- Navbar – ref
  - Contains <NavItem>
  - Automatic collapse into a hamburger icon
- … and many others