ECS 162
Web Programming

FAQ

- This week we are just working on the card creation screen. No need to add login or review pages.

- When the user hits enter in the text box on the left, translation should update.
- When the user clicks “save”, card should be stored in database.

Final design

React main ideas

- Create virutal HTML components to encapsulate widgets
- Programming illusion in which DOM is re-constructed, completely from scratch, whenever it changes – however you got into a particular UI state, that state always looks the same.
- What is the problem with reconstructing the DOM from scratch whenever anything changes?

Virtual DOM

- What is the drawback of this idea?
  - User does not want to see redraws, flashes, thing that they did not touch blinking, etc.
  - Slow!
  - Solution…
- Virtual DOM

Virtual DOM

- React maintains an internal copy of the DOM – this is the virtual DOM
- At every event:
  - Rebuilds virtual DOM from scratch
  - Compares to current real DOM
  - Makes minimal changes to get real DOM to match virtual DOM
- Surprisingly fast.

React Beginning

- HTML file is page with empty body. DOM will be constructed by React, not initialized by HTML

```
<body>
  <div id="root"></div>
  <script src="lango.js"></script>
</body>
```

- Can also combine DOM elements initialized in HTML with elements added by React

Load the React module in HTML

```
<script src="https://unpkg.com/react-
dom@16/umd/react-dom.development.js"></
script>
<script>src="https://unpkg.com/react@16/umd/rea
ct.development.js"></script>
```

React building the DOM

- In Javascript file,

```
const lango = React.createElement( 'h1', { id: 'logo' }, 'Lango!' );
```

- React.createElement returns a wrapper around a real <h1> HTML element.
- Arguments are: type of element, object containing inline properties, and contents.

React building the DOM

- To get the element onto the Web page, call:

```
ReactDOM.render(lango,
document.getElementById('root'));
```

- This tells React to add whatever is logo to the selected DOM element.

A virtual component

- Can define a component with a function:

```
function FirstCard() {
        return React.createElement(
                "div",
                { className: "textCard" }
            );
        }
```

- Rendering the component displays the div
- Components start with capital letters, real HTML elements with lowercase.

Virtual component with contents

```
function FirstCard() {
        return React.createElement(
                "div",
                { className: "textCard" },
                React.createElement(
                        "p", null, "Hello, world!" ) );
        }
```

- This could get messy if you wanted a lot of contents

JSX

- JSX is an extension to Javascript that is popular for working with the React module.

```
function FirstCard() {
        return (<div className="textCard">
        <p>Hello, world!</p>
        </div>);
        }
```

- Nicer way of saying what was on previous slide
- Those <> are not HTML, but are intended to look like HTML.

Getting JSX

- JSX is not built into Javascript and is not interpreted in the browser (in general…)
- You need to compile JSX into Javascript, and then use the resulting Javascript as usual
- There are various ways of doing this on your own machine. A popular approach:

  https://github.com/facebook/create-react-app
- I'll discuss how to do it minimally on the server.

Calling the compiler

npx babel lango.jsx --presets react-app/prod > lango.js

- Observe that result is what we expected
- Babel is the compiler
- npx is a command that runs executables in the node_modules directory
- Need to install them. Directions at the bottom of:

https://reactjs.org/docs/add-react-to-a-website.html

Making the compiler run itself

- Set the compiler to “watch” your JSX source code and when anything .jsx changes, recompile it automatically

```
npx babel --presets react-app/prod --extensions ".jsx"
. --out-dir . –watch &
```

- The & on the end makes the command run in the background
- Start this up each time you start working on your JSX

CSS and flexbox

- Layout and styling for these DOM elements is handled exactly as before – we still need CSS!

Combining in an element

```
const main = (<main>
            {lango}
            <FirstCard/>
           </main>);
```

- Parens just prevent the Javascript interpreter from helpfully inserting unwanted semicolons.
- lango is a variable, an expression that we want to evaluate. This is what the {} indicates.

The text entry card

```
function FirstInputCard() {
        return (<div className="textCard">
                <textarea />
                </div>);
          }
```

- Regular HTML textarea tag.
- Like the input tag but a bigger box.

Do something when user hits return

- We want to translate the text when the user hits the return key in the input box.
- So not an onclick…
- Use onKeyPress event, occurs every time user hits a key. Only actually do something (fire off AJAX request) when the key is the return key.
- How to attach event listeners to elements using React?

Adding onKeyPress function

```
function FirstInputCard () {
    return <div className="cardside">
        <input onKeyPress={checkReturn} />
        </div>;
}
```

- Basically same as adding it in HTML

Handling key press

```
function checkReturn(event) {
        console.log(event.charCode);
        }
```

- If the charCode is 13, it was the return key; fire off AJAX request and get translation