COMS/SE 319: Construction of User Interfaces
Spring 2021
LAB Activity 4 – React

Task 1: Setting up the environment.

In order to develop apps with React, we need to have Node.js installed. Please go to this link and download the latest version of Node.js based on your operating system.
https://nodejs.org/en/download/

1.1 Check all required packages are installed

Run the following command in Terminal/CMD.

```
node -v
```

```
(base) ali@Dell-Inspiron:~$ node -v
v12.18.3
```

Now let's make sure npm is also installed alongside Node.js.

```
npm -v
```

```
(base) ali@Dell-Inspiron:~$ npm -v
6.14.8
```

1.2 React and JSX

React creates elements (such as <p>, <button>, <input>) for web pages through a special syntax called JSX. JSX is a syntax extension to JavaScript.

JSX tags look similar to HTML tags, however they are much more powerful than HTML elements.

Open terminal and go to the folder 'LabActivity4' and run the following command to install the dependencies required for running the React app.

npm install

you will see this output:

```
added 394 packages from 175 contributors and audited 398 packages in 4.071s
```

Now run the following command in terminal and leave the terminal open:

npm run build_task1

Open '*LabActivity4/src/task_1.js*' in a text editor and let's add a simple paragraph to our page using JSX:

```
const para = <p>This is a paragraph created with JSX.</p>;

ReactDOM.render(para, document.getElementById("app"));
```

The paragraph element is assigned to *para* constant. Later, we call the *ReactDOM.render()* and pass the element as well as the container in the webpage indicating where we want the element to be rendered. In our case, we created a *'div'* element in *'/public/index.html'*, and passed that element as the container to the *ReactDOM.render()*.

***ReactDOM.render()* Should alway be at the last line in the js file.**

Save the JS file and open *'/public/index.html'* in the browser to see the results.

1.3 JSX, Elements and their children

JSX tags can have children. Moreover, it is also possible to use javascript expressions inside JSX tags.

Let's add a student object to our file and show the details of the student.

Comment all previous lines and add the following lines:

```
const student = {
    name: "Alex",
    class: "CS 319",
    year: 2020
};


const stud = (
    <div>
        <p>Student name: {student.name}</p>
        <p>Class name: {student.class}</p>
        <p>Year: {student.year}</p>
    </div>
);

ReactDOM.render(stud, document.getElementById("app"));
```

Notice how we use the ‘div’ tag and include multiple paragraph tags within the div element and assign them to the constant ‘stud’ which later is rendered. As you can see we can use JavaScript expressions inside curly braces.

Refresh the ‘/public/index.html’ in the browser to see the results.

Task 2: React Components

In React, a web page is basically composed of a number of components. Components are pieces of UI elements. Components provide lots of advantages. For instance you define a component once and you can use it numerous times throughout different web pages of your web application. It is also possible to pass down data between components. Consider the figure below, look how a web page can be constructed by combining different components. In this figure there is a root component called ‘App’ that includes all other components. In React, we always have one root component that we pass it to the ReactDOM.render().

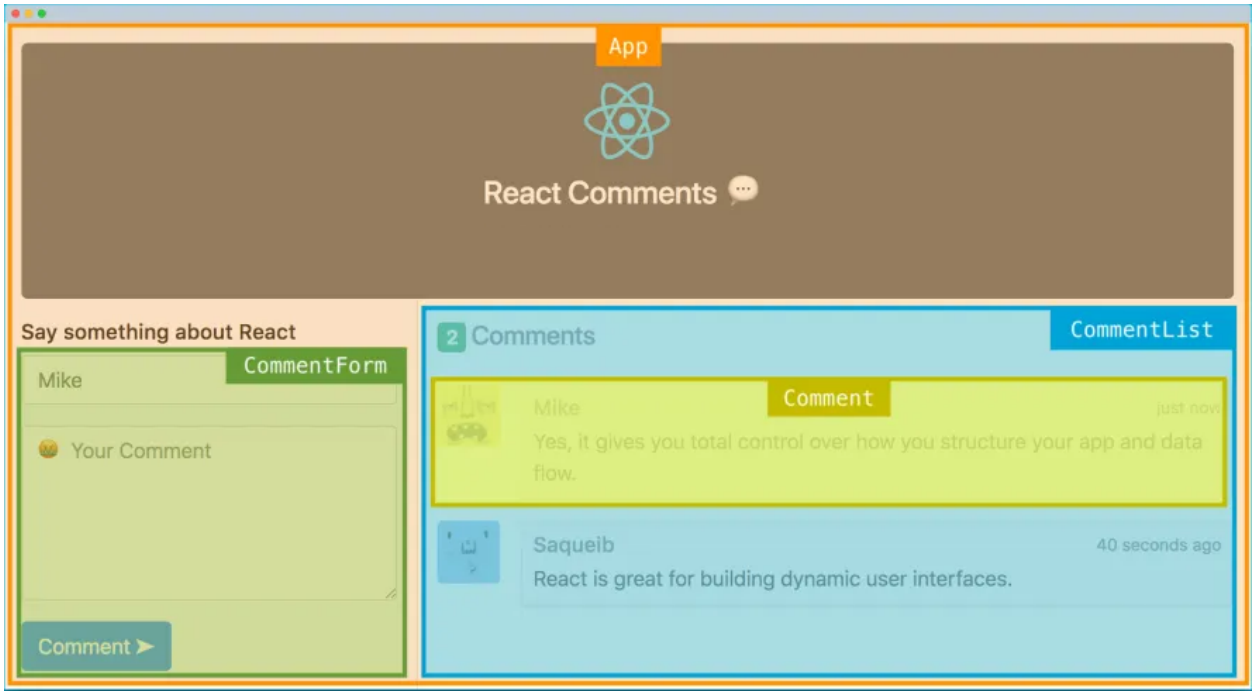

Image source: qcode.in

2.1 Creating our first component

In React, components can be created in two ways:

1. Defining a class as component (Class component)
2. Defining a function as component (Function component)

Let’s create our first component.

Go back to the terminal and terminate the last command by pressing ***control+c*** *on* Mac or ***ctrl+c*** in Windows.

Now run this command:

npm run build_task2

Keep the terminal open.

Open *'/LabAcitivy4/src/task_2.js'* in a text editor and add the following function component.

```
function App(){
    return(
        <div>
            <h3>Class: CS319</h3>
            <h3>Lab activity 4</h3>
            <p>This is function component.</p>
        </div>
    );
}

ReactDOM.render(<App />, document.getElementById("app"));
```

So we created a component named *'App'*. Notice how the component is passed to the ReactDOM.render(). We can reference a component as a JSX tag. Since we created a component with the name *'App'*, we use *<APP />* to reference the component. Component's name should always begin with a capital letter. Refresh *'LabActivity4/public/index.html'* in the browser to see the results.

Let's create the same component but this time we will use a class component.

```
class App extends React.Component{
    render(){
        return(
            <div>
            <h3>Class: CS319</h3>
            <h3>Lab activity 4</h3>
            <p>This is class component.</p>
        </div>
        );
    }
}

ReactDOM.render(<App />, document.getElementById("app"));
```

A class component should always inherit from *'React.Component'* class and implement the *'render()'* method.

Refresh *'LabActivity4/public/index.html'* in the browser, you will not see much difference, but now '*App*' is a class component.

2.2 Creating a layout using components

Let's build a simple website layout with React components.

Go back to terminal and stop the last command (*control+c* *on* Mac or *ctrl+c* in Windows), and run this command:

```
npm run build_task23
```

Now open *'LabActivity4/src/task_23.js'* in a text editor.

There are some predefined components in this file. Take your time and try to figure out what is the purpose of each component.

Go to the last line, create an *'App'* component there and add all other components as children of 'App'. Then call ReactDOM.render(). Like figure below:

```
class App extends React.Component{
    render(){
        return(
            <div>
            <Header />
            <Nav />
            <Article />
            <Footer />
            </div>
        );
    }
}


ReactDOM.render(<App />, document.getElementById("app"));
```

Refresh *'LabActivity4/public/index.html'* to see the output.

Task 3: Props and State

3.1 Props

In react, it is possible to pass data to child nodes through *props*. Consider the last example of the previous task, where we had a navigation menu on the left.

Think about a situation where we want to pass an array to the *'Nav'* component and *'Nav'* component shows the items of that array in the navigation menu.

For doing this we use a React concept called *props* to pass the array to the *'Nav'* component.

In order to send *props* to a component, we use the same syntax as HTML attributes.

```
<Nav props_name={props_value} />
```

In the component, we can access the props value via:

```
this.props.props_name
```

While task_23.js is open, Edit the *App component* to define an array and pass that array as *props* to the *Nav component.*

```
class App extends React.Component{

    render(){
        const menu_items = ['Introduction', 'Basics', 'JavaScript'];
        return(
            <div>
            <Header />
            <Nav menu_items={menu_items}/>
            <Article />
            <Footer />
            </div>
        );
    }
}
```

Now, edit the *Nav* component to show items of the array from props.

```
class Nav extends React.Component {
    render(){
        return(
            <nav>
                <ul>
                {this.props.menu_items.map(function(item){return <li key={item}>{item}</li>})}
                </ul>
            </nav>
        );
    }
}
```

Refresh the webpage to see the results. In fact, nothing will change, except that now the items in the navigation menu are being read from an array. You can add an item to the array and refresh the page to see the results.

3.2 State:

Remember in Task 2 we showed two different ways of defining React components; Class components and Function components. One of the differences between Class components and Function components is that Class components have an object called state where you can store property values that belong to the component. And whenever the state object is changed or updated, React will automatically rerender that component.

Now let's save the list of items of the navigation menu in the App component's state.

Open *task_23*.js in editor.

The state object should be defined in the constructor method.

So, modify the App component as follows:

```
class App extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            menu_items : ['Introduction', 'Basics', 'JavaScript']
        };
    }
```

And in the render method of the *App component,* we pass the *state object* as a *prop*.

```
render(){
    return(
        <div>
        <Header />
        <Nav menu_items={this.state.menu_items}/>
        <Article />
        <Footer />
        </div>
    );
}
```

If you refresh the page, you will see no difference, but now the items in the list are being read from the App component's state. Whenever the state object changes, the corresponding components will re-render automatically to present the latest version of the state. Let's see this in action.

We will add an input field to the web page so that we can add items to our menu, like the figure below:

In *task_23.js*, modify the render function of Nav component as follows:

```
class Nav extends React.Component {
    render() {
        return (
            <nav>
                <ul>
                    {this.props.menu_items.map(function (item) { return <li key={item}>{item}</li> })}
                </ul>
                <form>
                    <input type="text" name="new_item" />
                    <button>Add item</button>
                </form>
            </nav>
        );
    }
}
```

In the *App component* let's add a function before the render function. This new function will be responsible of receiving a text from the input field and appending that text to the this.state.menu_items.

We define this function as follows:

```
handleAddItem(event){
    event.preventDefault();
    const new_item = event.target.elements.new_item.value.trim();

    if (!new_item) {
        alert('Enter valid value to add item');
    }

    this.setState(function(prevState, props){
        return {
            menu_items: prevState.menu_items.concat(new_item)
        };
    });

}
```

This function will be triggered when we click on the *'Add item'* button. *'Add item'* button will call this function with the event object as the argument. We receive the event object as an argument and call

event.preventDefault() on it. This will prevent the default behaviours of submitting a form. After that we extract the value in the input field via *'event.target.elements.new_iem.value'*. And use this value to update the state. *This.setState* is used to update the state.

In the constructor of the *App component* we have to bind this *handleAddItem* function, so that this function will have access to the state object of the *App component*.

```
constructor(props){
    super(props);
    this.state = {
        menu_items : ['Introduction', 'Basics', 'JavaScript']
    };
    this.handleAddItem = this.handleAddItem.bind(this);
}
```

Now let's pass this *'handleAddItem'* function as a *prop* to the *'Nav' component*, so that the *'Nav' component* can call this function whenever the button is clicked.

Modify the render function of App component as follows:

```
render() {
    return (
        <div>
            <Header />
            <Nav menu_items={this.state.menu_items} handleAddItem={this.handleAddItem} />
            <Article />
            <Footer />
        </div>
    );
}
```

Modify *'Nav'* component as follow to use the props:

```
class Nav extends React.Component {
    render() {
        return (
            <nav>
                <ul>
                    {this.props.menu_items.map(function (item) { return <li key={item}>{item}</li> })}
                </ul>
                <form onSubmit={this.props.handleAddItem}>
                    <input type="text" name="new_item" />
                    <button>Add item</button>
                </form>
            </nav>
        );
    }
}
```

Now refresh the web page. You should be able to add items to the *'menu_items'* list. Notice that you immediately see the new value added to the list.

Please remember to answer the questions in the quiz on Canvas.