National University of Computer & Emerging Sciences, Karachi
Computer Science Department

Course Code: SE-3003	Course : Web Engineering Lab

Spring 2023, Lab Manual – 11

LLO 02: Web application using PHP and MySQL

Contents:

Introduction to Web Engineering

Web Engineering is the application of systematic and quantifiable approaches (concepts methods, techniques tools) to cost - effective requirements analysis, design, implementation, testing, operation, and maintenance of **high quality Web applications.**

Technologies to be studied

- HTML
- CSS
- JavaScript
- Bootstrap
- JQuery
- PHP
- MySQL [Database]
- Laravel [PHP FRAMEWORK]
- REACT JS

Tools – IDEs

- Visual Studio Code

- Adobe Dreamweaver
- Visual Studio
- XAMPP

10.1 Introduction to REACT:

React is a UI library developed at Facebook to facilitate the creation of interactive, stateful & reusable UI components. As Wikipedia puts it, React allows developers to "create large web-applications that use data and can change over time without reloading the page."

When JavaScript interacts with an HTML document, it is interacting with the Document Object Model, or DOM, which is a tree containing all of the HTML elements of that document. React uses a concept called the Virtual DOM to selectively render nodes on this tree. In essence, this allows React to interact with as little of the DOM as possible while still adequately making changes to the state of a webpage.

As an example, let's pretend that your website is the full body rendering of a person, from head to toe. The DOM would say "on the FACE, there are the EYES, NOSE, EARS, etc.". But, what happens if you want to change a feature on the person's body, like give the person some beautiful Steve Buscemi eyes. Normally, the person will completely be re-rendered with the new eyes. However, in React-land, when we apply these changes, two things take place. First, React runs a "diffing" algorithm, which identifies what has changed. The second step is reconciliation, where it updates the DOM with the results of the diff (i.e. changing the eyes to Steve's).

If the idea of the Virtual DOM is confusing, don't worry! It's difficult to wrap one's head around, and knowing exactly how the DOM and Virtual DOM function aren't necessary for this lab. Just know that React is used because it optimizes DOM interaction.

We have already studied DOM in previous lab, to know more about DOM is and how it functions, thislink is very helpful.

10.2 Getting Started:

1. Check that you have node installed by running `node --version` in your terminal
2. If you don't have node you can install it here; get the appropriate version for your computer
3. In cmd, run `npm install -g create-react-app`
4. Once everything is downloaded, move into the directory in which you want your app to be created (`cd Documents`)
5. Run `create-react-app my-app` to create your app
6. Then run `cd my-app` to go into that directory
7. Run `npm start` to start your app
8. In your browser, go to http://localhost:3000/ to see your app live!
9. Now open up any text editor of your choice and open the my-app folder
10. All of the changes that we will be making will be in the src directory

10.3 The Basics:

React's basic building blocks are called components. Let's write one. Create a new file in the `src` directory called `HelloWorld.jsx`. Add this code inside:

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


class HelloWorld extends Component {render() {
    return (
        <h1>Hello World!</h1>
    }
}
```

Change `App.js` to look like this:

```
import React, { Component } from 'react';import
'./App.css';
import HelloWorld from './HelloWorld';

class App extends Component {
    render() {
        return (
            <div className="App">
                <HelloWorld />
            </div>
        );
```

After saving your changes, you can view them in your localhost tab without needing to re-run from the command line.

If you haven't seen this syntax before, you are probably wondering what Javascript/HTML chimera sorcery is taking place right now.

10.4 Introduction to JSX:

This so-called sorcery is called JSX, and it is a Javascript XML syntax transform. This lets you write HTML-*ish* tags in your Javascript. Note that this is not exact HTML—you are really just writing XML-based object representations.

For regular html tags, the class attribute is `className` and the for attribute is `htmlFor` in JSX because these are reserved words in Javascript. A more in-depth explanation of JSX can be found here. While you can certainly use React without JSX, we highly recommend that you use JSX.

10.5 REACT Components:

React components are independent, reusable classes that compose different parts of your UI. React applications are made up of components, many of which are rendered within other components.

It is generally a good idea to write each component in its own file. For example, the `HelloWorld` component is implemented in `HelloWorld.jsx`. To use it elsewhere, simply import the component at the top of the file that you wish to use it in, as we did at the top of `App.js`. Note that you must export the component at the bottom of its own file in order to be able to import it elsewhere.

10.6 REACT Props:

When we use our defined components, we can add attributes called props that are passed from parent components. These attributes are available in our components as `this.props` and can be used to render dynamic data. In the `render()` method of `App.js`, add the following and change `[YOUR NAME]` to your name:

```
render() {
  return (

    <div className="App">

      <HelloWorld name={'[YOUR NAME]'} />

    </div>
```

In `HelloWorld.jsx`, add the following to the `render`() method:

```
render() {
  return (

    <h1>Hello, {this.props.name}!</h1>

  );
```

In this example, we added a name prop to the `HelloWorld` component, which we passed to the component in `App.js`.

Lifecycle Methods

The `render()` method is the only required method for creating a component, but there are several lifecycle methods and specs we can use that can be helpful which you can read about here.

State:

Every component has a state object and a props object. Initial state should be set in the `constructor()`, but can be set or reset elsewhere using the `setState()` method. Calling `setState()` triggers UI updates and is the bread and butter of React's interactivity. Let's create try this by implementing a Counter component in a new file, `Counter.jsx`:

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


class Counter extends Component {
    constructor(props) {

        super(props);


        this.state = {
            count: 5

        };

    }


    render() {
        return (

            <h1>{this.state.count}</h1>

        );

    }

}


export default Counter;
```

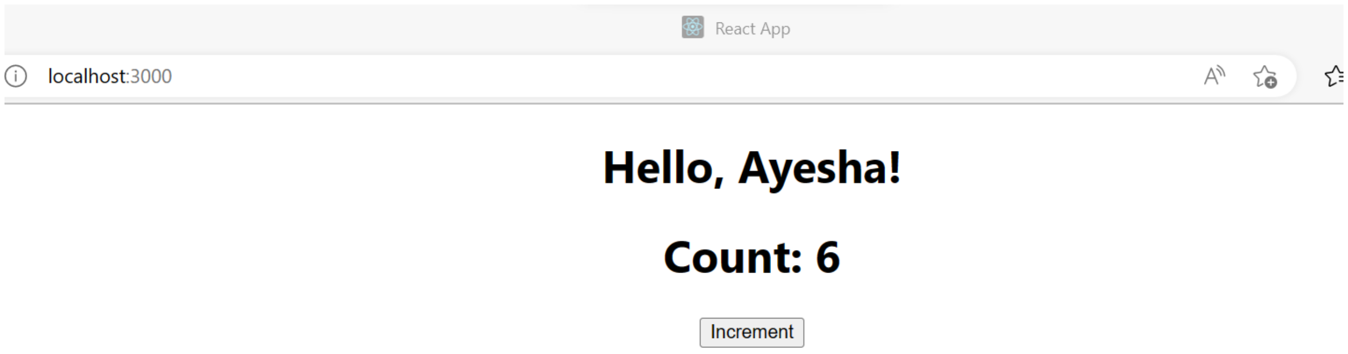
In `App.js`, add a `Counter` component beneath your `HelloWorld` component. Don't forget to import the Counter at the top of the file!

Events:

React also has a built in cross browser events system. The events are attached as properties of components and can trigger methods. Let’s make our count increment below using events. You can use `this.setstate()` to modify state.

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


class Counter extends Component { constructor(props) {

super(props);


this.state = { count: 5

};

}


  incrementCount = () => {

    this.setState({

      count: this.state.count + 1

    });

  }


  render() {

    return (

      <div>

        <h1>Count: {this.state.count}</h1>

        <button onClick={this.incrementCount}>Increment</button>

      </div>


);

}

}

export default Counter;
```

Once you’ve completed your Counter, check through browser. It should something like this.



10.7 Unilateral Data Flow:

In React, application data flows unidirectionally via the state and props objects, as opposed to the two-way binding of libraries like Angular. This means that, in a multi-component hierarchy, a common parent component should manage the stateand pass it down to child components via props.

To ensure that a UI refresh will occur if necessary, always update your state using the `setState()` method rather than by mutating `this.state` directly. The resulting values can be passed down to child components using attributes that are accessible via this.props. See this example below that shows this concept in practice. Go ahead and copy/paste it to see it live!

Here is an overview of what is happening below:

App.js passes a list of produce into an instance of FilteredList by a prop and then renders this instance of FilteredList onto the screen. This FilteredList is a componentthat adds an input field to the webpage that will filter the list of produce. Each time auser changes the inputted text in the search bar, it changes the 'search' state in FilteredList. FilteredList also has a child component called List (List renders the filtered produce list onto the webpage), which we pass as a prop the filtered list of produce(the list of produce comes from App.js, and the list is filtered in filterItem() based on the search state--the text in the search bar). Whenever the search state is changed by a user (i.e. changing the text in the search bar), the list of filtered items that is passed to List changes, and so the list of filtered produce on the webpage changes.

In `App.js`:

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

 import './App.css';

import FilteredList from './FilteredList';


/*

This list of produce that is passed into the FilteredList component. Notice that it is a
list of javascript objects where {key: value}.

*/

const produce = [

{name: "Apple", type: "Fruit"},

{name: "Pineapple", type: "Fruit"},

{name: "Banana", type: "Fruit"},

{name: "Pear", type: "Fruit"},

{name: "Strawberry", type: "Fruit"},

{name: "Orange", type: "Fruit"},

{name: "Lettuce", type: "Vegetable"},

{name: "Cucumber", type: "Vegetable"},

{name: "Eggplant", type: "Vegetable"},

{name: "Squash", type: "Vegetable"},

{name: "Bell pepper", type: "Vegetable"},

{name: "Onion", type: "Vegetable"},

];


class App extends Component { render() {

return (

<div className="App">

{/*

The list of produce will be passed into the FilteredList component the items property.

*/}

<FilteredList items={produce} />

</div>

);

}

}
```

```
export default App;
```

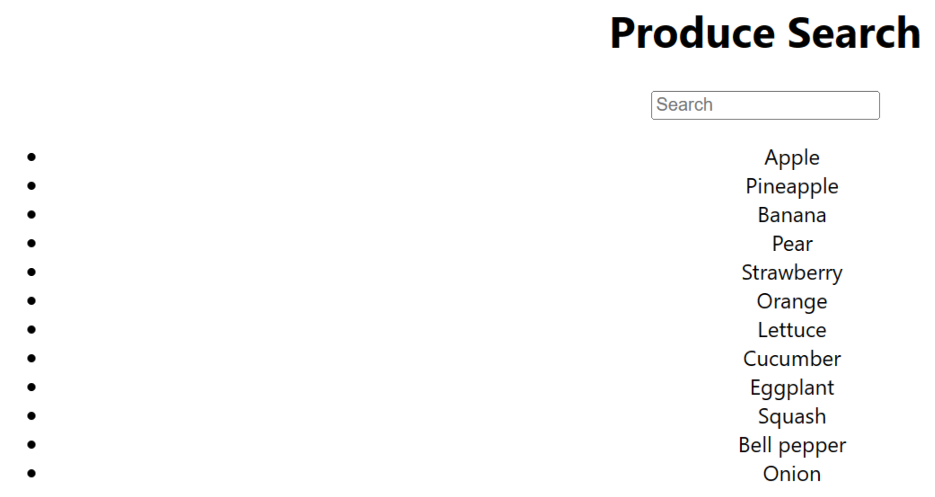
Create a new file called `FilteredList.jsx` and paste the following:

```
import React, { Component } from 'react';
import List from './List';
class FilteredList extends Component {
constructor(props) {
super(props);
// The state is just a list of key/value pair (like a hashmap)
this.state = {
search: ""
};
}
// Sets the state whenever the user types on the search bar
onSearch = (event) => {
this.setState({search: event.target.value.toLowerCase()});
}
filterItem = (item) => {
// Checks if the current search term is contained in this item
return item.name.toLowerCase().search(this.state.search) !== -1;
}
render() {
return (
<div className="filter-list">
<h1>Produce Search</h1>
<input type="text" placeholder="Search" onChange={this.onSearch} />
{/*
Here we are taking the items property (which is the list of
produce), filtering the content to match the search word, then
passing the filtered produce into the List component.
*/}
<List items={this.props.items.filter(this.filterItem)} />
</div>
);
}
}
export default FilteredList;
```

Create a new file called `List.jsx` and paste the following:

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

/*
The list component will take the list of items passed in as a property
and create an HTML list with those item. In this example, we are passing in the filtered
produce list, but this component can be used for other types of items as long as it has a
name.
*/
class List extends Component { renderList() {
/*
Javascript map will let you iterate and modify each item in a list.
In this example, we are changing each item
(ex. {name: "Apple", type: "Fruit"}) into a HTML list element.
*/
const items = this.props.items.map(item => { return <li key={item.name}>{item.name}</li>
});

return items;
}
render() { return (
<ul>
{this.renderList()}
</ul>
);
}
}

export default List;
```

Final Browser Look should be like this:



10.8 Task: Create a Dropdown Button Filter:

Now that we have reviewed some React basics, let's try adding a dropdown button! Since we will be using Bootstrap's dropdown menu, start by installing Bootstrap. From the my-app folder in your terminal, run `npm install react-bootstrap --save`

Add the following css files in `my-app/public/index.html` for styling:

```
<link    rel="stylesheet"    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
```

Your task will be to add in a dropdown menu that will filter out produce by type (fruit and vegetables). When fruit is selected, only fruit produce should show and when vegetable is selected, only vegetable produce should show on the list. You will also have to make sure that the dropdown will work with the search filter so that the list will only show produce that fulfills both the search and dropdown filters. We have provided you with an updated FilteredList.jsx with some hints on how to implement below. You will also need to look at the documentation on react-bootstrap dropdown to fill in a todo.

In `FilteredList.jsx:`

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

import { DropdownButton, MenuItem} from 'react-bootstrap';
import List from './List';


class FilteredList extends Component {
  constructor(props) {

    super(props);

    // TODO: Add a new key/value pair in the state to keep track of typethis.state
    = {

      search: ""

    };

  }
  // Sets the state whenever the user types on the search baronSearch =
  (event) => {

    this.setState({search: event.target.value.trim().toLowerCase()});

  }


  filterItem = (item) => {

    // Checks if the current search term is contained in this item

    // TODO: Add condition to check item's type

    return item.name.toLowerCase().search(this.state.search) !== -1;

  }

  /* TODO: Add an event handling method for when an item in dropdown is selected

    Per the DropdownButton documentation, this function should take in an eventKey andevent

  */
  render() {

    return (

        <div className="filter-list">

           <h1>Produce Search</h1>

        {/* TODO: Add more menu items with onSelect handlers*/}

        <DropdownButton id="typeDropdown" title={"Type"}>

          <MenuItem eventKey="all" onSelect={HANDLER FUNCTION HERE}>All</MenuItem>

        </DropdownButton>

        <input type="text" placeholder="Search" onChange={this.onSearch} />

        <List items={this.props.items.filter(this.filterItem)} />

      </div>

    );

  }
}

export default FilteredList;
```

Make sure all three of your filter selections work - including the “All”dropdown - in order to fully complete your Dropdown Button Filter.

Styling (Optional):

If you finished early, you can go ahead and change `App.css` to style your filtered list!