Please note that you will need to generate a UR for the parts mentioned. Generate a link for each via GitHub pages. you can do this last once you have completed all tasks. Go to the bottom of the document and see the details on how to submit. About React ? Quick Re-cap Facebook created the UI library React to make it easier to create interactive, stateful, and reusable UI components. React enables programmers to "create large web applications that use data and can change over time without reloading the page," according to Wikipedia. The Document Object Model, also known as DOM, is a tree that houses all of the HTML elements in an HTML document with which JavaScript communicates. To render certain nodes of this tree in a specific order, React uses a concept known as the Virtual DOM. In essence, this enables React to interact with the DOM as little as possible while still effectively altering a webpage's state. 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. If you’re interested in learning more about what the DOM is and how it functions this link is very helpful. (Skip this if you have done the install) Let’s get 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 then go to terminal step 3 3. In your terminal, run sudo npm install -g create-react-app (you may need to type in your system password); note that Windows users should instead run npm install -g create-react app, as Windows does not have the “‘sudo” command 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 The Guide Code Guide code for the files you are required to add/change for this assignment can be found here. Although working with the stencil code is not a requirement for the lab, many students have found it easier to use these files rather than trying to copy and paste the code provided in the Assignment. In order to use these files, simply download them from mysite via codd shown above and copy them into the src directory in the application directory you created just now. 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: Change App.js to look like this: 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. About 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. The 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. The 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: 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. The 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 (

{this.state.count}

); } } 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! The 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. To get an idea of how to add a button you can look at Facebook’s documentation here. import React, { Component } from 'react'; class Counter extends Component { constructor(props) { super(props); this.state = { count: 5 }; } incrementCount = () => { /\* TODO: Use setState() to modify the count. Here’s an example: this.setState({ keyToChange: valueToChangeTo }); \*/ } render() { return (

{this.state.count}

{/\* TODO: add a button that calls incrementCount() when clicked \*/}

); } } export default Counter; Once you’ve completed your Counter, save it and create a url to point to this show you have done this!! (Yup Publish this – Github pages) The 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 state and 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 component that adds an input field to the webpage that will filter the list of produce. Each time a user 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. What’s 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 (

{/\* The list of produce will be passed into the FilteredList component the items property. \*/}

); } } export default App; Create a new file called FilteredList.jsx and paste the following: Now you will 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

 {item.name} }); return items; } render() { return (

{this.renderList()}

); } } export default List; Final 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: The new task will be to add 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 products that fulfill 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. 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 type this.state = { search: "" }; } In FilteredList.jsx: // Sets the state whenever the user types on the search bar onSearch (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 and event \*/ render() { return (

Produce Search

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

); } } export default FilteredList; Make sure all three of your filter selections work - including the “All” dropdown - to fully complete your Dropdown Button Filter. Once you’ve completed your Filter, save it and create a URL to point to this show you have done this!! See below how to generate the link Additional work - Styling - You must create your own custom styling and yes it MUST be well-designed. Additional Resources Facebook’s official React tutorial React video tutorial React documentation Facebook talk explaining the rationale behind using React React Developer Tools (Chrome, Firefox) This assignment was adapted from: https://scotch.io/tutorials/learning-react-getting-started-and concepts How to submit: Publishing a basic React app using GitHub Pages is a straightforward process. Generate the link and upload to the i-college CW folder 1. Set Up Your React Application: o Create a new React application using create-react-app. Open your terminal and navigate to your preferred directory: o cd desktop o npx create-react-app "your-project-name" o This will set up a new React app in the specified directory. 2. Create a GitHub Repository: o Go to your GitHub account and create a new repository for your project. o After creating the repository, you’ll see a page with options. Leave it open for now. 3. Push Your React App to the Repository: o Initialize Git in your project directory: o git init o Add your GitHub repository as a remote: o git remote add origin git@github.com:username/myproject.git o Commit your changes and push them to the repository: o git add. o git commit -m "Initial commit" o git push -u origin main 4. Configure GitHub Pages: o In your project’s repository on GitHub, click the Settings tab. o Scroll down to the GitHub Pages section. o Under Source, select the branch you want to publish (usually main or master). o Click Save. GitHub will display the URL for your project. 5. Deploy Your React App: o Generate a production build of your app: o npm run build o Deploy your code to GitHub Pages: o npm run deploy 6. Verify Your Deployment: o Visit the GitHub Pages link (e.g., https://username.github.io/repositoryname) to see your React app live!