React – Inner Components

In this lab, you will create a React application in a Kraken app.

# Objectives

In this lab, you will

* Use an existing Kraken app named, ‘react-app’
* Create React components from an existing HTML page
* Pass data to child components
* Pass data from the child back to the parent component
* Run the app

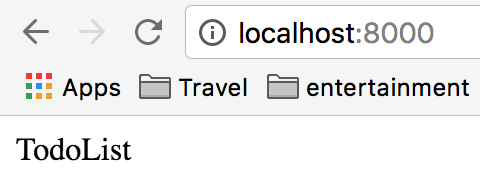
# Run the Kraken app to verify correctness

1. Change to the lab/react-app folder and run

npm install

npm start

1. Open the browser to localhost:8000 and the page should look like this:



1. The output comes from public/views/layout.jsx as shown.

**'use strict'**;  
  
**var *React*** = ***require***(**'react'**);  
**var *Link*** = ***require***(**'react-router'**).**Link**;  
  
***module***.exports = ***React***.**createClass**({  
  
 render: **function** *render*() {  
  
 **return** (  
 <**html**>  
 <**head**>  
 <**meta charSet='utf-8'** />  
 <**title**>  
 {**this**.props.**title**}  
 </**title**>  
 <**link rel='stylesheet' href='/css/styles.css'** />  
 </**head**>  
 <**body**>  
 {**this**.props.children}  
 <**script src='/bundle.js'**></**script**>  
 </**body**>  
 </**html**>  
 );  
 }  
});

1. This displays the page content. It includes all the children components defined in the /routes.jsx file as shown below:

**'use strict'**;  
  
**var *React*** = ***require***(**'react'**);  
**var *ReactRouter*** = ***require***(**'react-router'**);  
**var *Router*** = ***ReactRouter***.**Router**;  
**var *Route*** = ***ReactRouter***.**Route**;  
**var *IndexRoute*** = ***ReactRouter***.**IndexRoute**;  
  
**var *Layout*** = ***require***(**'./public/views/layout.jsx'**);  
**var *Todos*** = ***require***(**'./public/views/todos.jsx'**);  
  
**var *routes*** = ***module***.exports = (  
 <**Router**>  
 <**Route path='/' component=**{***Layout***}>  
 <**IndexRoute component=**{***Todos***} />  
 </**Route**>  
 </**Router**>  
);

1. Notice the <IndexRoute> tag includes the /public/views/todos.jsx which defines the <TodoList> component as shown:

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
  
 render: **function** *render*() {  
  
 **return** (  
 <**div**>TodoList</**div**>  
 )  
 }  
});

1. The component just renders the string, TodoList, which appears on the browser page.

# Add the HTML code

1. Copy the contents of the <body> tag in /public/todos.html into the render() method as shown above. Replace the <div>TodoList</div>.
2. Edit the text, replacing all attributes name, class, with the name, className. Class is a reserved word in ES6 and can’t be used in JSX, so JSX renamed it.
3. Stop and start the application. The browser should display the Todo List just like the html file.

# Refactor the HTML into Components

1. Move the <form> tag and its contents to the TodoInput render() method. Move the <div className=”item”> tag and its contents to the TodoItem render() method.
2. Reference the child components in TodoList as shown below:

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
  
 render: **function** *render*() {  
 **return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput** />  
   
 <**TodoItem** />  
  
 </**div**>  
 </**div**>  
 )  
 }  
});

1. Stop and restart the application. The browser should display the same page after refreshing.

# Communication with Children

1. Let’s fix the communication with the <TodoItem> first.
2. The <TodoItem> must display one todo object so let’s pass the todo item from the parent to the child as shown:

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
  
 render: **function** *render*() {  
 **var** todo = **'First'**;  
  
 **return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput** />  
  
 <**TodoItem todo=**{todo} />  
  
 </**div**>  
 </**div**>  
 )  
 }  
});

1. Edit the TodoItem. Remove the ‘href’ attribute and display the input data from the property, ‘todo’.

**var *TodoItem*** = ***React***.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div className="item"**>  
 <**a  
 title="Update this todo item"  
 className="update-link"**> {this.props.**todo**} </**a**>  
 <**a  
 title="Delete this todo item"  
 className="del-btn"**>Delete</**a**>  
 </**div**>  
 );  
 }  
});

1. Notice the child refers to the input attribute with {this.props.todo}. Later, we will modify the todo data type to be an object with two properties, id and content.
2. Go back to the parent and add more todo items for display.
3. Edit TodoList object. Add a method, getInitialState(), and return the initial state. Add an initial state property, todos, as an array of strings. Edit the render() method to display all the items.

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
 getInitialState() {  
 **return** {  
 **todos**: [**'First'**, **'Second'**]  
 }  
 },  
  
 render: **function** *render*() {  
  
 **return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput** />  
  
 {**this**.**state**.**todos**.**map**( **function**(todo, idx) {  
 **return** ( <**TodoItem key=**{idx} **todo=**{todo} /> );  
 })}  
  
 </**div**>  
 </**div**>  
 )  
 }  
});

1. When rendering the list of <TodoItem>s, notice the key parameter. This allows react to have unique id’s for each element, making refresh more efficient since it only renders the components that have changed.
2. Restart the server and refresh the browser. There should now be TWO todo items.

# Fix the Delete Button

1. First let’s change the data type of each todo to be an object with id and content properties.
2. Edit the TodoList to change the todo type. NOTE: I also added the index property used later.

**var *TodoList*** = ***module***.exports = ***React***.**createClass**({  
 getInitialState() {  
 **return** {  
 **todos**: [  
 {**id**: **'1'**, **content**: **'First'**},  
 {**id**: **'2'**, **content**: **'Second'**}  
 ],  
 **index**: 3  
 }  
 },

1. Edit the TodoItem to display the todo.content instead of the todo.

**var *TodoItem*** = ***React***.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div className="item"**>  
 <**a  
 title="Update this todo item"  
 className="update-link"**> {**this**.**props**.**todo**.**content**} </**a**>  
 <**a  
 title="Delete this todo item"  
 className="del-btn"**>Delete</**a**>  
 </**div**>  
 );  
 }  
});

1. Restart the app and refresh the page. Verify that the page has not changed.
2. Create two event handlers for the two <a> tags. The first one should be named update() and the second, delete(). Each should have only one parameter, event.

**var *TodoItem*** = ***React***.**createClass**({  
 render: **function** () {  
 **return** (  
 <**div className="item"**>  
 <**a data-todoid=**{**this**.**props**.**todo**.**id**} **onClick=**{**this**.update}  
 **title="Update this todo item"  
 className="update-link"**> {**this**.**props**.**todo**.**content**} </**a**>  
 <**a data-todoid=**{**this**.**props**.**todo**.**id**} **onClick=**{**this**.delete}  
 **title="Delete this todo item"  
 className="del-btn"**>Delete</**a**>  
 </**div**>  
 );  
 },  
 update(event) {  
 **var** id = event.**target**.getAttribute(**'data-todoid'**);  
 alert( id );  
 },  
 **delete**(event) {  
 **var** id = event.**target**.getAttribute(**'data-todoid'**);  
 alert( id );  
 }

}

1. Notice in the render() method, we added the attribute, data-todoid, which contains the todo.id value. In the event handler, we get the value with event.target.getAttribute(‘data-todoid’)
2. Also, notice the attribute name ‘data-\*’. All the ‘data-‘ attributes must be in LOWER CASE.
3. Restart the app and refresh the browser. When you click on the todo item text OR the delete button, you should see the id.

# Send Data Back to the Parent

1. Now, let’s send the data back to the parent so it can delete the element by using the id.
2. Edit the parent, TodoList, and add a handler to delete the item by id and pass the handler to the child using the delete={this.delete}.

render: **function** *render*() {

**var** self = **this**;

**return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput** />  
  
 {**this**.**state**.**todos**.**map**( **function**(todo, idx) {  
 **return** ( <**TodoItem key=**{idx} **todo=**{todo} **delete=**{**self**.delete} /> );  
 })}  
  
 </**div**>  
 </**div**>  
 )  
},  
**delete**(id) {  
 **var** todos = **this**.state.**todos**.filter( **function**(todo) {  
 **return** todo.**id** != id  
 });  
 **this**.**setState**( { **todos**: todos });  
}

1. Notice the delete() method removes the item from the list and then calls this.setState() which tells React to re-render the component because the state changed.
2. In the child, TodoItem, invoke this callback in the delete() event handler.

update(event) {  
 **var** id = event.**target**.getAttribute(**'data-todoId'**);  
 alert( id );  
},  
**delete**(event) {  
 **var** id = event.**target**.getAttribute(**'data-todoId'**);  
 **this**.**props**.delete( id );  
}

1. Notice the callback is passed via an attribute in the Parent and referenced as a property in the child.
2. Restart the app, refresh the browser, and delete a todo item.

# Fix the TodoInput Component

1. In the parent, add a method called, addTodo( content ). Pass this callback to the <TodoInput> component using the addTodo attribute.

getInitialState() {  
 **return** {  
 **todos**: [  
 {**id**: **'1'**, **content**: **'First'**},  
 {**id**: **'2'**, **content**: **'Second'**}  
 ],  
 **index**: 3  
 }  
},  
  
render: **function** *render*() {  
 **var** self = **this**;  
  
 **return** (  
 <**div id="layout"**>  
 <**h1 id="page-title"**>Express Todo</**h1**>  
 <**div id="list"**>  
  
 <**TodoInput addTodo=**{**this**.addTodo} />  
  
 {**this**.**state**.**todos**.**map**( **function**(todo, idx) {  
 **return** ( <**TodoItem key=**{idx} **todo=**{todo} **delete=**{self.delete} /> );  
 })}  
  
 </**div**>  
 </**div**>  
 )  
},  
addTodo( content ) {  
 **var** todo = { **id**: **this**.state.**index**++, **content**: content };  
 **this**.**setState**( { **todos**: **this**.state.**todos**.concat( todo ) } );  
  
},

1. Note that I added a property to the state, index. I use this value to create the id for new todo items. Also, notice the setState() method to force React to update the component.
2. In the TodoInput, add an submit( event ) event handler to the form, onSubmit, attribute.

**var *TodoInput*** = ***React***.**createClass**({  
 render: **function** () {  
 **return** (  
 <**form onSubmit=**{**this**.submit}>  
 <**div className="item-new"**>  
 <**input type="text" name="content" className="input"**/>  
 </**div**>  
 </**form**>  
 );  
 },  
 submit( event ) {  
 event.preventDefault();  
 **var** content = event.**target**.elements[0].**value**;  
 event.**target**.elements[0].**value** = **''**;  
 **this**.**props**.addTodo( content );  
 }  
});

1. In the event handler, if the target is a form, the elements property is an array of input elements inside the form. Here we access the first input element and extract the value. Then we reset the value to a blank string.
2. Restart the app, refresh the page, and add some todo items.

Congratulations. You have completed this lab.