

Steps to creating the React-To-Do app:

Since the typical React app has multiple components, instead of structuring inside of HTML document will divide components into multiple files and then import/export them to each other as needed.

Such a structure requires a build tool such as Webpack, Grunt, or Gulp to enable this functionality. The build tool also transpiles(converts) JSX to a version of javascript which can be rendered by most browsers. For apps using build tools, source files are typically contained in a `src` folder while the `public` or `dist` folders hold files generated by the build tool.

Rather than manually configuring the React app and creating its initial boilerplate, and rather than manually configuring Webpack, will use Node Package Manager (npm) and the `create-react-app` command-line tool to create the initial build of the project.

1. install node.js and npm:

Download installer at node.js website

2. verify installation at command line:

```
node -v
```

3. install the `create-react-app` tool at command line:

```
npm install -g create-react-app
```

4. Move to the desired directory, and then perform the initial build of the react-to-do app:

```
create-react-app react-to-do
```

Notice structure of the build:

- `node_modules` directory holds app's dependencies. By default added to `git.ignore` file so not part of git

repository

- `public` directory holds bundles created by Webpack. DevServer accesses/retrieves files from here
- `src` directory holds original source code. Webpack creates its bundles from these files
- `src/index.js` : root javascript file that imports React and renders root component of the application, called `App.js`
- `src/App.js` : the root component of the React application. All other components of application are descendants of `App.js`

5. Move to the `react-to-do` directory, and spin-up the DevServer in inline mode so you can monitor app behavior as you code the app:

```
>cd react-to-do  
>npm start
```

6. Go to `App.js`, delete the default boilerplate in the return statement of the `render()` method of this component. Can also delete the `import` statement pertaining to the logo
7. Within `src` directory, create a `components` directory:

```
>cd src  
>mkdir components
```

8. move to `components` directory, and create a new file `ToDo` for holding the React component:

```
>cd components
```

```
>touch ToDo.js
```

9. go to `ToDo.js` and create the component. 4 steps: importing React and React's component class, creating a React class component, adding a `render()` method to the component, and then exporting the class. Remember that the `render()` function must return only one root element, uses JSX syntax which is transpiled by Webpack to javascript, and must use `className` rather than `class` to declare class attributes:

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

```
class ToDo extends Component{  
  render(){  
    return(  
      <li>A todo will go here</li>  
    );  
  }  
}
```

```
export default ToDo;
```

10. go back to `App.js` and import the `ToDo.js` component; place the import statement along with the other import statements at the top of the document:

```
import ToDo from './components/ToDo.js';
```

11. render the `ToDo` component by accessing it in the `App.js` `render()` method:

```
import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  render() {
    return (
      <div className="App">
        <ul>
          <ToDo />
          <ToDo />
        </ul>
      </div>
    );
  }
}

export default App;
```

12. add the constructor method to `App.js` component:

```
import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';
```

```

class App extends Component {
  constructor(props){
    super(props);
  }
  render() {
    return (
      <div className="App">
        <ul>
          <ToDo />
          <ToDo />
        </ul>
      </div>
    );
  }
}

export default App;

```

13. define initial state for `App.js` by declaring an array to `ToDo`'s. Each element within the array is an object with two key-value pairs: description and boolean for whether the task is completed:

```

import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

```

```

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ]
    );
  }
  render() {
    return (
      <div className="App">
        <ul>
          <ToDo />
          <ToDo />
        </ul>
      </div>
    );
  }
}

export default App;

```

14. Display the ToDo's by iterating through the `todo` array within

the `render()` method of `App.js`. Use the `.map()` for the iteration as this creates a new array without mutating the original array:

```
import React, { Component } from 'react';
import './App.css';
import Todo from './components/Todo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ]
    );
  }
  render() {
    return (
      <div className="App">
        <ul>
          { this.state.todos.map( (todo, index) =>
            <ToDo
              key={ index }
            />
          )}
```

```

        )
      }
    </ul>
  </div>

  );
}
}

export default App;

```

15. Pass down the `description` and `isCompleted` properties of the `todos` state from the `App.js` component to the `ToDo.js` component. Remember when rendering `ToDo` from within the `.map()` function the assignment values are in curly braces, and in terms of the parameters defined for the `.map()` function:

```

import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false

```



```

    }
    ]
    );
  }
  render() {
    return (
      <div className="App">
        <ul>
          { this.state.todos.map( (todo, index) =>
            <ToDo
              key={ index }
              description = {todo.description}
              isCompleted = {todo.isCompleted}
            />
          )
        }
      </ul>
    </div>
    );
  }
}

export default App;

```

- Go to `ToDo.js` and change its `render()` function so it can access the props passed down to it by `App.js` :

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

```

class ToDo extends Component {
  render() {
    return (
      <li>{ this.props.description }</li>
    );
  }
}

export default ToDo;

```

17. Further change `ToDo` component so that it renders not only the `todo` description but also a checkbox which is checked when the task is completed:

```

import React, { Component } from 'react';

class ToDo extends Component {
  render() {
    return (
      <li>
        <input type='checkbox' checked={this.props.isCompleted} />
        <span>{this.props.description}</span>
      </li>
    );
  }
}

```

```
export default ToDo;
```

18. Add functionality to the app such that changes to the checkbox status in the `ToDo` component can be transmitted to the `App` and state of the `isCompleted` property can be updated. Do this by creating an event handler in `App` component and passing it down to `ToDo` as a prop. You cannot add the event handler method to the `ToDo` component as data hierarchy travels in one direction downward, so you wouldn't be able to update state in `App` with this approach. Create the event handler function `toggleComplete` in `App` component and have it print 'toggleComplete executed' to the console. Also pass the `toggleComplete` function to `ToDo` as a prop:

```
import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted:
false},
      {description: 'Buy new dishes', isCompleted: false}
    ]
```

```
    );  
  }
```

```
  toggleComplete(){  
    console.log('toggleComplete executed');  
  }
```

```
  render() {  
    return (  
      <div className="App">  
        <ul>  
          { this.state.todos.map( (todo, index) =>  
            <ToDo  
              key={ index }  
              description = {todo.description}  
              isCompleted = {todo.isCompleted}  
              toggleComplete = {this.ToggleComplete}  
            />  
          )  
        }  
      </ul>  
    </div>  
  );  
}  
}
```

```
export default App;
```

19. add an event listener `onChange` to the `ToDo` component's checkbox input. When there is a change in status of the checkbox, invoke the parent component `App.js` `toggleComplete` event handler function:

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

class ToDo extends Component {
  render() {
    return (
      <li>
        <input
          type='checkbox'
          checked={this.props.isCompleted}
          onChange={this.props.toggleComplete}
        />
        <span>{this.props.description}</span>
      </li>
    );
  }
}

export default ToDo;
```

20. Change the `toggleComplete` function in `App` component so it is an anonymous function so it can take the `index` from the `.map()` function as a parameter. Add the `index` as a parameter to the function declaration as well:

```
import React, { Component } from 'react';
import './App.css';
import Todo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted:
false},
      {description: 'Buy new dishes', isCompleted: false}
    ]
    );
  }

  toggleComplete(index){
    console.log('toggleComplete executed');
  }

  render() {
    return (
      <div className="App">
        <ul>
          { this.state.todos.map( (todo, index) =>
            <ToDo
              key={ index }

```

```

        description = {todo.description}
        isCompleted = {todo.isCompleted}
        toggleComplete = {() => this.toggleComplete(ind
ex)}}
      />
    )
  }
</ul>
</div>
);
}
}

export default App;

```

21. Write the `toggleComplete()` function so that it updates the state of `isCompleted` property within the `App` component. Do NOT mutate the state `todos` array. Instead, copy the array first using the `.slice()` method, then modify the value of this array copy appropriately, and finally use the `setState()` method to assign the modified, copied array to the new state. Don't forget the `this` keyword before the `setState()` method:

```

import React, { Component } from 'react';
import './App.css';
import Todo from './components/ToDo.js';

class App extends Component {

```

```
constructor(props){
```

```
  super(props);
```

```
  this.state=(todos:[
```

```
    {description: 'Walk the cat', isCompleted: true},
```

```
    {description: 'Throw the dishes away', isCompleted  
: false},
```

```
    {description: 'Buy new dishes', isCompleted: false  
}
```

```
  ]
```

```
);
```

```
}
```

```
toggleComplete(index){
```

```
  const todos = this.state.todos.slice();
```

```
  const todo = todos[index];
```

```
  todo.isCompleted = todo.isCompleted ? false : true  
;
```

```
  this.setState({todos: todos});
```

```
}
```

```
render() {
```

```
  return (
```

```
    <div className="App">
```

```
      <ul>
```

```
        { this.state.todos.map( (todo, index) =>
```

```
          <ToDo
```

```
            key={ index }
```

```
            description = {todo.description}
```



```

        isCompleted = {todo.isCompleted}
        toggleComplete = {() => this.toggleComplete(in
dex)}}
    />
  )
  }
</ul>
</div>
);
}
}

export default App;

```

22. We want to add a form to the application so the user can add a new ToDo item. Have `App.js` render this form. Create an event listener `onSubmit` for when the user clicks the `submit` button, and have this listener invoke a `handleSubmit()` event-handler function. Use an arrow expression for this function so that `this` is preserved so that manual binding using the `.bind()` function is unnecessary:

```

import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  constructor(props){

```

```
    super(props);  
    this.state=(todos:[  
      {description: 'Walk the cat', isCompleted: true},  
      {description: 'Throw the dishes away', isCompleted  
: false},  
      {description: 'Buy new dishes', isCompleted: false  
}  
    ]  
    );  
  }
```

```
  toggleComplete(index){  
    const todos = this.state.todos.slice();  
    const todo = todos[index];  
    todo.isCompleted = todo.isCompleted ? false : true  
;  
    this.setState({todos: todos});  
  }
```

```
  render() {  
    return (  
      <div className="App">  
        <ul>  
          { this.state.todos.map( (todo, index) =>  
            <ToDo  
              key={ index }  
              description = {todo.description}  
              isCompleted = {todo.isCompleted}
```

```

        toggleComplete = {() => this.toggleComplete(in
dex)}}
    />
  )
  }
</ul>
<form onSubmit={ (e) => this.handleSubmit(e)}>
  <input type='text' />
  <input type='submit' />
</form>
</div>
);
}
}

export default App;

```

23. construct the `handleSubmit()` function. Have it prevent default execution of a page reload upon the form submitting its data, and console log when the submit button is pressed to verify it is working properly:

```

import React, { Component } from 'react';
import './App.css';
import Todo from './components/Todo.js';

class App extends Component {
  constructor(props){

```

```
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ]
    );
  }
```

```
toggleComplete(index){
  const todos = this.state.todos.slice();
  const todo = todos[index];
  todo.isCompleted = todo.isCompleted ? false : true
;
  this.setState({todos: todos});
}
```

```
handleSubmit(e){
  e.preventDefault();
  console.log('handleSubmit called');
}
```

```
render() {
  return (
    <div className="App">
      <ul>
```

```

        { this.state.todos.map( (todo, index) =>
        <ToDo
        key={ index }
        description = {todo.description}
        isCompleted = {todo.isCompleted}
        toggleComplete = {() => this.toggleComplete(index)}
        />
        )
        }
    </ul>
    <form onSubmit={(e)=> this.handleSubmit(e)}>
        <input type='text' />
        <input type='submit' />
    </form>
</div>
);
}
}

export default App;

```

24. Update the state of `App` when user is creating a new `ToDo` in the text input by adding a `newToDoDescription` property to `App.js` state:

```

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

```

```
import Todo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ],
    newToDoDescription:''
    );
  }

  toggleComplete(index){
    const todos = this.state.todos.slice();
    const todo = todos[index];
    todo.isCompleted = todo.isCompleted ? false : true
;
    this.setState({todos: todos});
  }

  handleSubmit(e){
    e.preventDefault();
    console.log('handleSubmit called');
  }
}
```

```
render() {
  return (
    <div className="App">
      <ul>
        { this.state.todos.map( (todo, index) =>
          <ToDo
            key={ index }
            description = {todo.description}
            isCompleted = {todo.isCompleted}
            toggleComplete = {( ) => this.toggleComplete(index)}
          />
        )
      }
    </ul>
    <form onSubmit={(e)=> this.handleSubmit(e)}>
      <input type='text' />
      <input type='submit' />
    </form>
  </div>
  );
}

export default App;
```

25. Now assign the value of the text input in the `App` component to the `newToDoDescription` state property of `App.js`. Also add an event listener that invokes a new function called `handleChange()`, using an arrow expression:

```
import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ],
    newToDoDescription:''
    );
  }

  toggleComplete(index){
    const todos = this.state.todos.slice();
    const todo = todos[index];
    todo.isCompleted = todo.isCompleted ? false : true
;
  }
}
```



```
        this.setState({todos: todos});
    }

    handleSubmit(e){
        e.preventDefault();
        console.log('handleSubmit called');
    }

    render() {
        return (
            <div className="App">
                <ul>
                    { this.state.todos.map( (todo, index) =>
                        <ToDo
                            key={ index }
                            description = {todo.description}
                            isCompleted = {todo.isCompleted}
                            toggleComplete = {( ) => this.toggleComplete(index)}
                        />
                    )
                }
            </ul>
            <form onSubmit={(e)=> this.handleSubmit(e)}>
                <input type='text'
                    value={this.state.newToDoDescription}
                    onChange={(e) => this.handleChange(e)}
                />
            </form>
        )
    }
}
```

```

        <input type='submit' />
      </form>
    </div>
  );
}
}

export default App;

```

26. Write the `handleChange()` function, using the `e` or `event` parameter to update the state of the `newToDoDescription` state property to the value inside the text input:

```

import React, { Component } from 'react';
import './App.css';
import ToDo from './components/ToDo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ],
    newToDoDescription:''

```

```
);  
}
```

```
toggleComplete(index){  
  const todos = this.state.todos.slice();  
  const todo = todos[index];  
  todo.isCompleted = todo.isCompleted ? false : true  
;  
  this.setState({todos: todos});  
}
```

```
handleSubmit(e){  
  e.preventDefault();  
  console.log('handleSubmit called');  
}
```

```
handleChange(e){  
  this.setState({newToDoDescription: e.target.value}  
)  
}
```

```
render() {  
  return (  
    <div className="App">  
      <ul>  
        { this.state.todos.map( (todo, index) =>  
          <ToDo  
            key={ index }  
          </ToDo>  
        ) }  
      </ul>  
    </div>  
  )  
}
```

```

        description = {todo.description}
        isCompleted = {todo.isCompleted}
        toggleComplete = {() => this.toggleComplete(index)}
      />
    )
  }
</ul>
<form onSubmit={(e)=> this.handleSubmit(e)}>
  <input type='text'
    value={this.state.newToDoDescription}
    onChange={(e) => this.handleChange(e)}
  />
  <input type='submit' />
</form>
</div>
);
}
}

export default App;

```

27. Go back to the `handleSubmit()` function and use it to update the state of the `todos` state array with the new ToDo item upon the user clicking the `submit` button. Use Javascript spread syntax so that the old `todos` array is not mutated:

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

```
import './App.css';

import Todo from './components/Todo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ],
    newToDoDescription:''
  );
}

  toggleComplete(index){
    const todos = this.state.todos.slice();
    const todo = todos[index];
    todo.isCompleted = todo.isCompleted ? false : true
;
    this.setState({todos: todos});
  }

  handleSubmit(e){
    e.preventDefault();
    const newToDo={description: this.state.newToDoDesc
```

```
ription, isCompleted: false};

    this.setState({todos: [...this.state.todos, newToDo] });
  }

  handleChange(e){
    this.setState({newToDoDescription: e.target.value})
  }

  render() {
    return (
      <div className="App">
        <ul>
          { this.state.todos.map( (todo, index) =>
            <ToDo
              key={ index }
              description = {todo.description}
              isCompleted = {todo.isCompleted}
              toggleComplete = {( ) => this.toggleComplete(index)}
            />
          )
        }
        </ul>
        <form onSubmit={(e)=> this.handleSubmit(e)}>
          <input type='text'
            value={this.state.newToDoDescription}
          />
        </form>
      </div>
    )
  }
}
```

```

        onChange={(e) => this.handleChange(e)}
      />
      <input type='submit' />
    </form>
  </div>
);
}
}

export default App;

```

28. Prevent the application from allowing the user to submit an empty Todo by adjusting the `handleSubmit()` function in 2 ways. First, after updating state the `newToDoDescription` property is reset to a value of an empty string. Secondly, add a statement that if the text input value is empty, that the function stops (use the `return` keyword without an argument to accomplish this):

```

import React, { Component } from 'react';
import './App.css';
import Todo from './components/Todo.js';

class App extends Component {
  constructor(props){
    super(props);
    this.state=(todos:[
      {description: 'Walk the cat', isCompleted: true},

```

```
      {description: 'Throw the dishes away', isCompleted
: false},
      {description: 'Buy new dishes', isCompleted: false
}
    ],
    newToDoDescription: ''
  );
}
```

```
toggleComplete(index){
  const todos = this.state.todos.slice();
  const todo = todos[index];
  todo.isCompleted = todo.isCompleted ? false : true
;
  this.setState({todos: todos});
}
```

```
handleSubmit(e){
  e.preventDefault();
  if(!this.state.newToDoDescription){return}
  const newToDo={description: this.state.newToDoDesc
ription, isCompleted: false};
  this.setState({todos: [...this.state.todos, newToD
o], newToDoDescription: ''});
}
```

```
handleChange(e){
  this.setState({newToDoDescription: e.target.value}
```



```
)  
  
}  
  
render() {  
  return (  
    <div className="App">  
      <ul>  
        { this.state.todos.map( (todo, index) =>  
          <ToDo  
            key={ index }  
            description = {todo.description}  
            isCompleted = {todo.isCompleted}  
            toggleComplete = {( ) => this.toggleComplete(in  
dex)}  
          />  
        )  
      }  
    </ul>  
    <form onSubmit={(e)=> this.handleSubmit(e)}>  
      <input type='text'  
        value={this.state.newToDoDescription}  
        onChange={(e) => this.handleChange(e)}  
      />  
      <input type='submit' />  
    </form>  
  </div>  
);  
}
```

```
}
```

```
export default App;
```