## The Backend Server

The Contacts app project that we're building is a front-end project. However, we'll eventually be storing the contacts on a backend server. Since we're only really focusing on the front-end for this course, we've gone ahead and built this server for you so you can focus on just the React parts of this program.

The server is just a simple Node/Express app. The repo for the project is at [**https://github.com/udacity/reactnd-contacts-server2**](https://github.com/udacity/reactnd-contacts-server2). All you need to do is:

* clone the project with git clone https://github.com/udacity/reactnd-contacts-server2.git
* install the project dependencies with npm install
* start the server with node server.js

Once you've started the server, you can forget about it. The Contacts project we're working on will interact with this server, but we won't ever modify any of the server code.

## 💡 Running Two Servers💡

At this point, you should be running two different servers on your local machine:

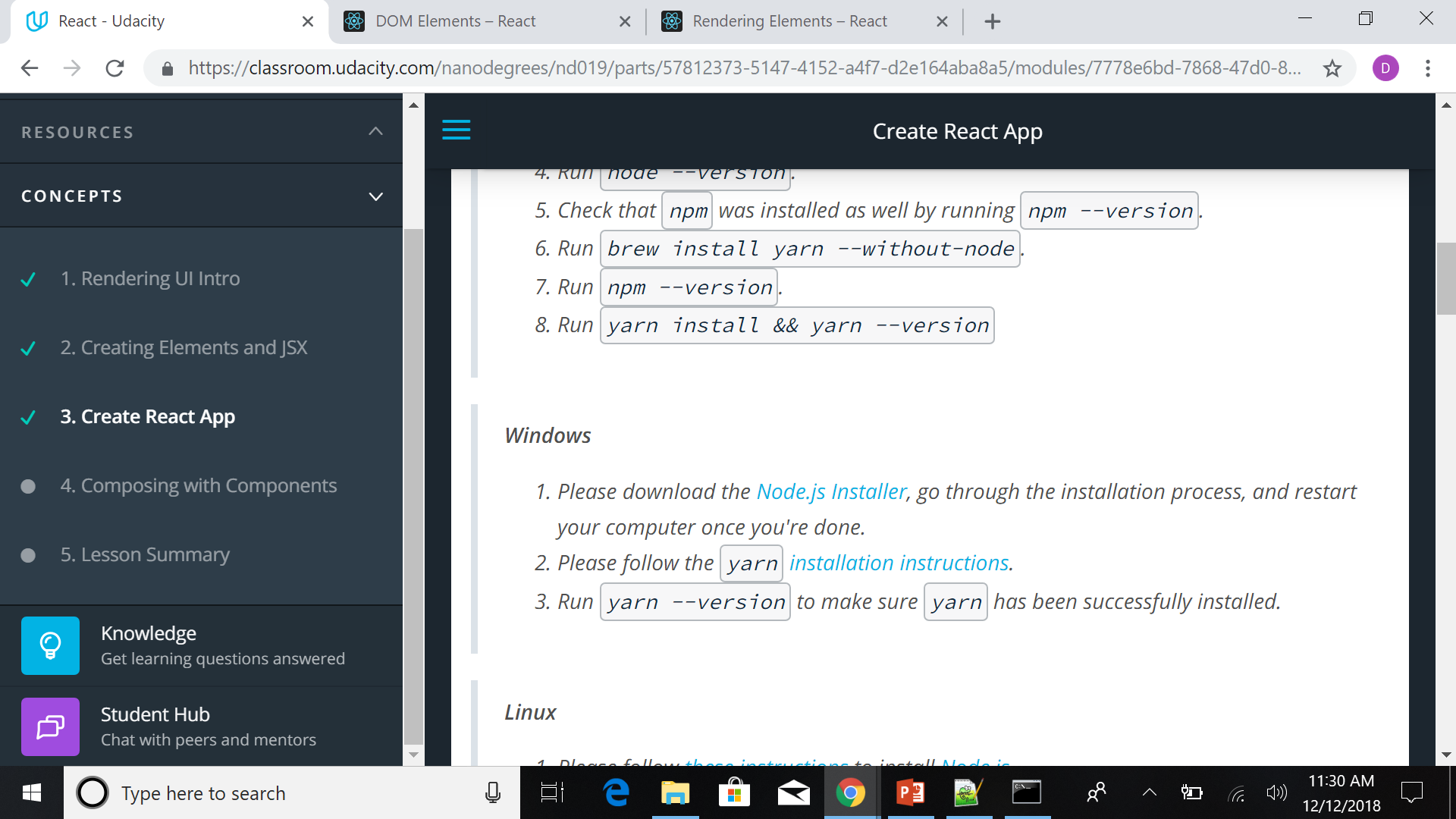
* Front-end development server: Accessible on **port 3000** (with npm start or yarn start)
* Back-end server: Accessible on **port 5001** (with node server.js)

Please be sure that both are running before moving on in this Lesson.

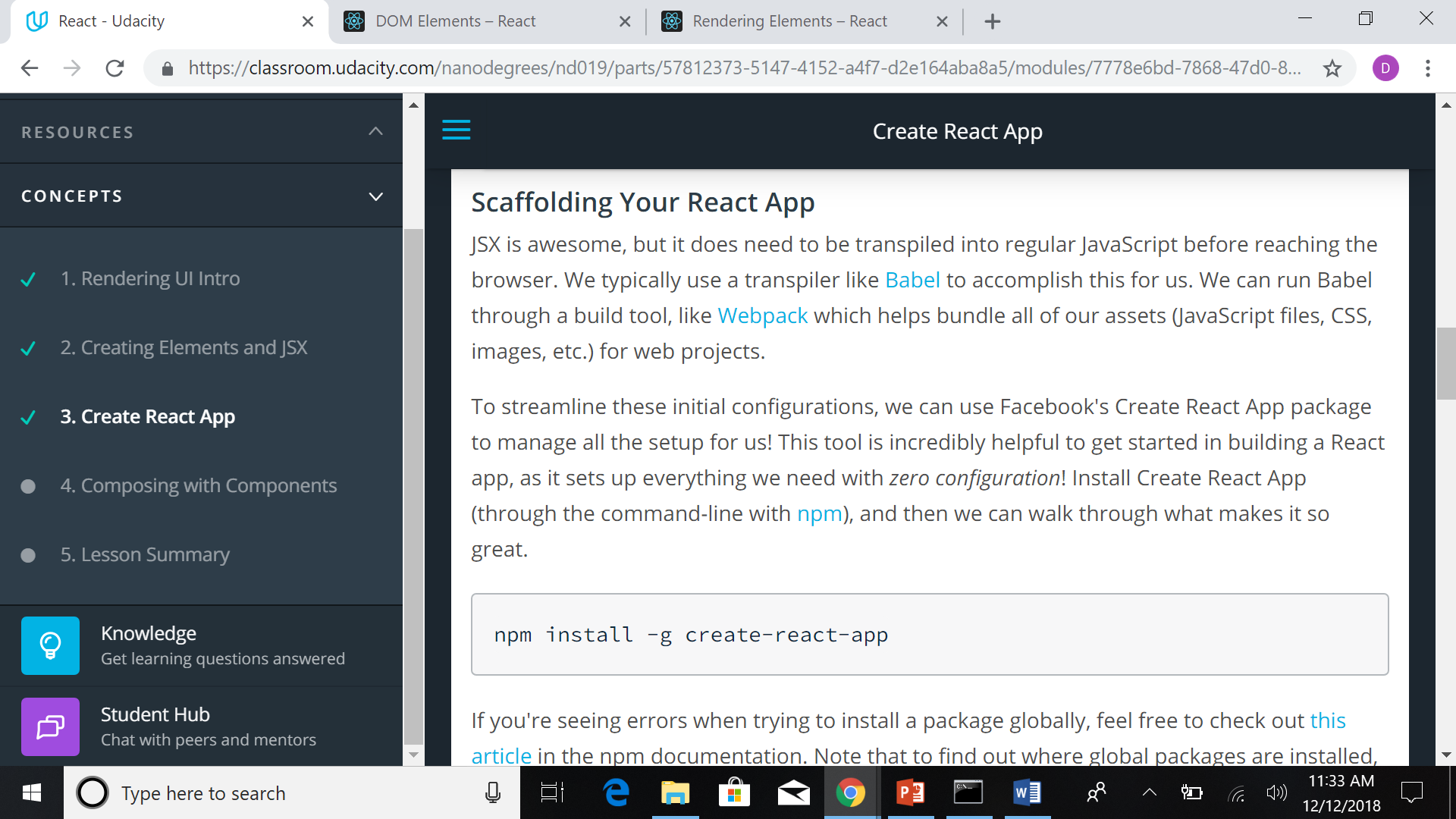
Creating react application

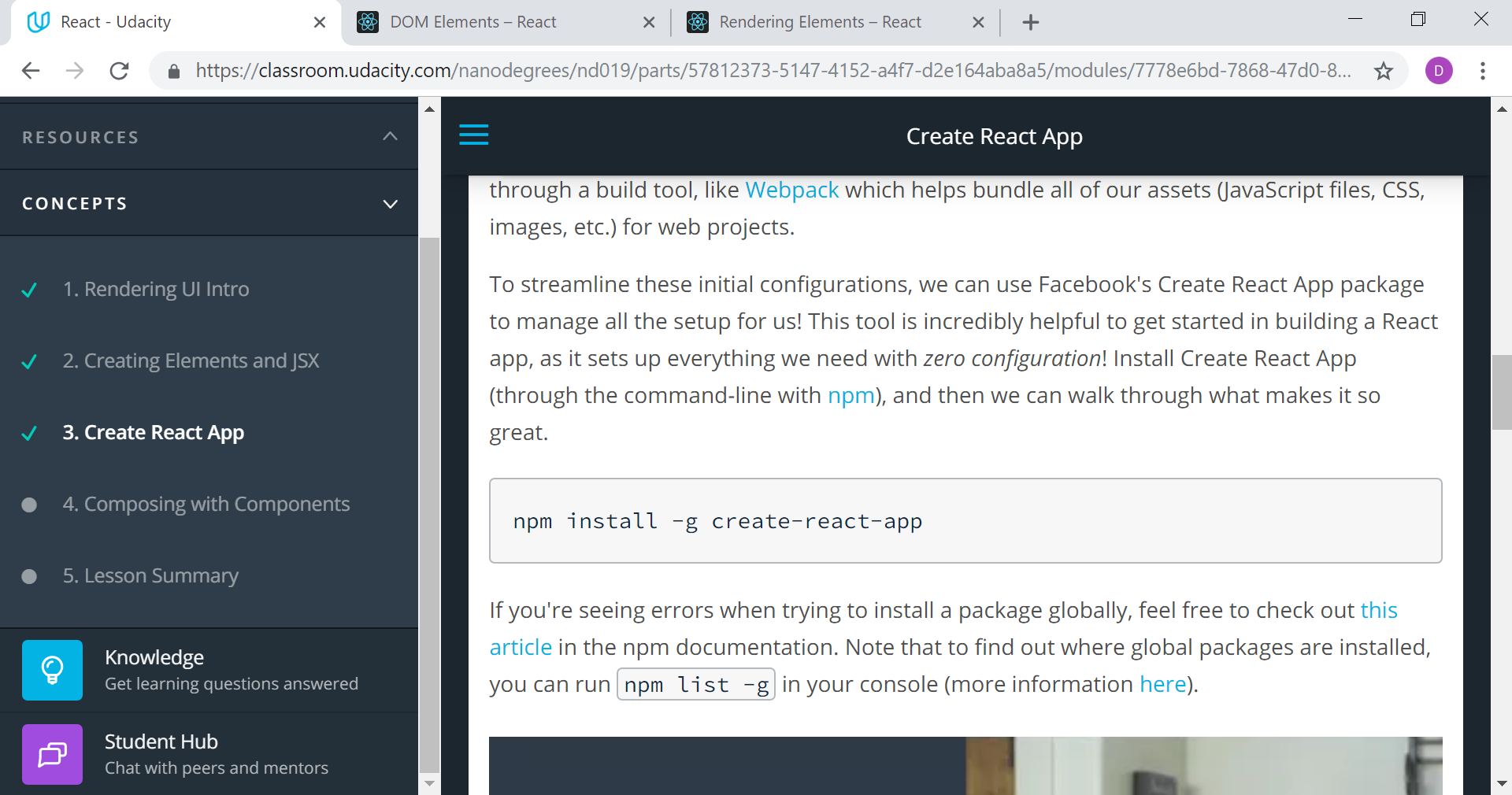
Npx create-react-app <application-name>

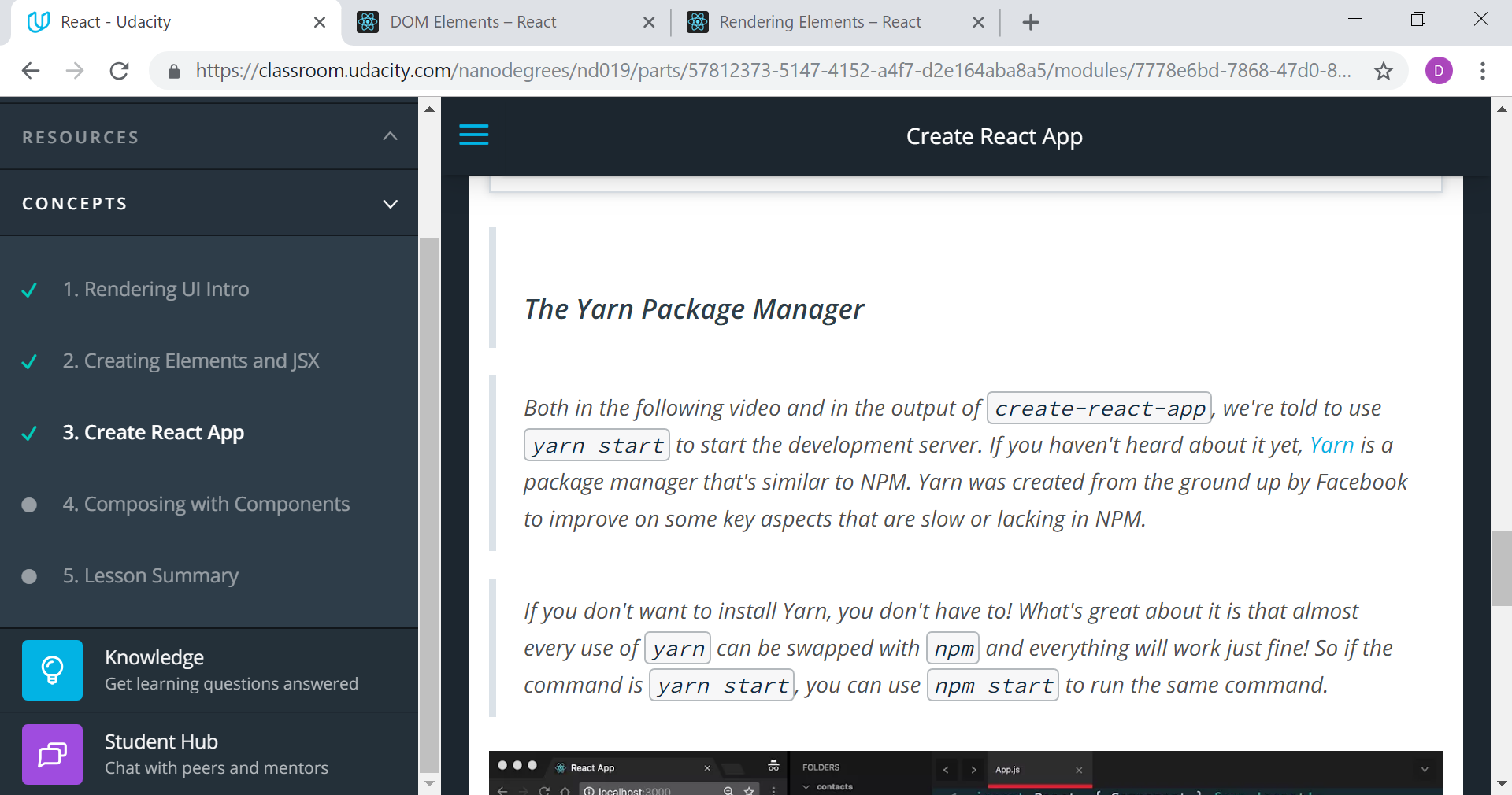
Installing React on Windows

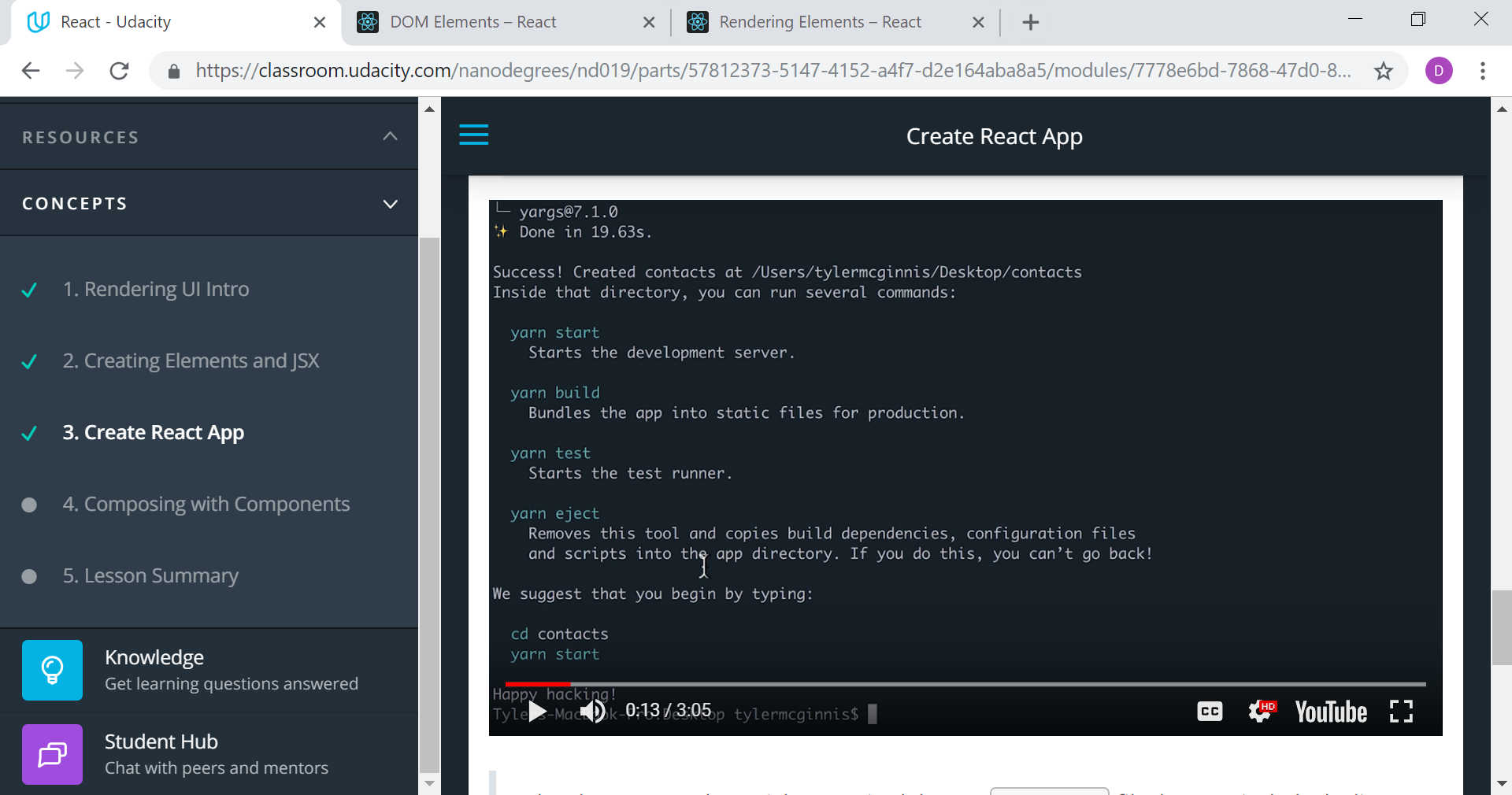


Install Babel, WebPack using Facebook’s React-App









[**Here's the commit with the changes made in this video.**](https://github.com/udacity/reactnd-contacts-app/commit/df432ab6b51202e23382487cea26ecf2276dce76)

Note that the value attribute is set on the <input> element. Since the displayed value will always be the value in the component's state, we can treat state, then, as the "single source of truth" for the form's state.

To recap how user input affects the ListContacts component's own state:

1. The user enters text into the input field.
2. The onChange event listener invokes the updateQuery() function.
3. updateQuery() then calls setState(), merging in the new state to update the component's internal state.
4. Because its state has changed, the ListContacts component re-renders.

Let's see how we can leverage this updated state to filter our contacts. To help us with our filtering we'll need the following packages:

* [**escape-string-regexp**](https://www.npmjs.com/package/escape-string-regexp)
* [**sort-by**](https://www.npmjs.com/package/sort-by)
* npm install --save escape-string-regexp sort-by

**Type checking a Component's Props with PropTypes**

As we implement additional features into our app, we may soon find ourselves debugging our components more frequently. For example, what if the props that we pass to our components end up being an unintended data type (e.g. an object instead of an array)? PropTypes is a package that lets us define the data type we want to see right from the get-go and warn us during development if the prop that's passed to the component doesn't match what is expected.

To use PropTypes in our app, we need to install [**prop-types**](https://facebook.github.io/react/docs/typechecking-with-proptypes.html):

npm install --save prop-types

Alternatively, if you have been using [**yarn**](https://www.npmjs.com/package/yarn) to manage packages, feel free to use it as well to install:

yarn add prop-types

Property Types

**import** PropTypes **from** 'prop-types';

|  |
| --- |
|  |
| ListConts.propTypes = { |
|  | contacts: PropTypes.array.isRequired, |
|  | onDeleteContact: PropTypes.func.isRequired, |
|  | }  Guard operator in javascript   |  | | --- | | {showingContacts.length !== contacts.length ***&&*** ( | |  |  |  | <div className='showing-contacts'> | |  |  |  | <span>Now showing {showingContacts.length} of {contacts.length}</span> | |  |  |  | <button onClick={this.clearQuery}>Show all</button> | |  |  |  | </div> | |  |  |  | )} | |
|  |  |

Lifecycle events of component:

To recap, lifecycle events are special methods that React provides that allow us to hook into different points in a component's life to run some code. Now, there are a number of different lifecycle events. They will run at different points, but we can break them down into three distinct categories:

### Adding to the DOM

The following lifecycle events will be called in order when a component is being added to the DOM:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

⚠️componentWillMount() has been deprecated. ⚠️

As of React 16.3, componentWillMount() has been replaced with UNSAFE\_componentWillMount(). Only UNSAFE\_componentWillMount() will work starting with React 17.0. UNSAFE\_componentWillMount() is now considered to be a legacy method and should not be used in new code.

### Re-rendering

The following lifecycle events will be called in order when a component is re-rendered to the DOM:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()[**(specific use cases)**](https://reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
5. componentDidUpdate()

⚠️componentWillReceiveProps() and componentWillUpdate() have been deprecated. ⚠️

As of React 16.3, they have been replaced with UNSAFE\_componentWillUpdate() and UNSAFE\_componentWillReceiveProps(). Only UNSAFE\_componentWillUpdate() and UNSAFE\_componentWillReceiveProps() will work starting with React 17.0. UNSAFE\_componentWillUpdate() and UNSAFE\_componentWillReceiveProps() are now considered to be legacy methods and should not be used in new code.

### Removing from the DOM

This lifecycle event is called when a component is being removed from the DOM:

* componentWillUnmount()

### Further Research

* [**componentDidMount()**](https://facebook.github.io/react/docs/react-component.html#componentdidmount) from the React Docs
* [**componentWillUnmount()**](https://facebook.github.io/react/docs/react-component.html#componentwillunmount) from the React Docs
* [**Component Lifecycles**](https://facebook.github.io/react/docs/react-component.html#the-component-lifecycle) from the React Docs

## Install React Router

To use React Router in our app, we need to install [**react-router-dom**](https://www.npmjs.com/package/react-router-dom).

npm install --save react-router-dom

Here is the code straight from the React Router repository.

**class** **BrowserRouter** **extends** **React**.**Component** {

static propTypes = {

basename: PropTypes.string,

forceRefresh: PropTypes.bool,

getUserConfirmation: PropTypes.func,

keyLength: PropTypes.number,

children: PropTypes.node

}

history = createHistory(**this**.props)

render() {

**return** <Router history={this.history} children={this.props.children} />

}

}

React router has three components:

* 1. BrowserRouter
  2. Route
  3. Link
* *// using ES6 modules*
* **import** { BrowserRouter, Route, Link } **from** 'react-router-dom'