The course entails some of the features mentioned below

React

ReactStrap

Redux

Fetch

The node environment is intertwined with Git and that's one of the prime reasons why it's good to be well versed in Git

To intitialize a folder in your workspace as a git repository just go to the directory in the powershell and type git init.

This marks the particular folder as a master branch

git status will give you the basic details of the repository

git add .

you will add all the files in the folder to the in the staging area

after this write

git commit -m "some message"

which will take a screenshot of the state and will be saved in your memory

git log --oneline will give you a brief history of your commits

merely typing git log will give you a much verbose display of all the commit detials

The next git command that we're going to learn about is git checkout. So this checkout command allows us to checkout a filefrom a previous commit in our git repository. So if we don't like the current file that we have in our folder, and we want to go back to a previous version of the file. We can always check out the file from a previous commit, or from the current commit, and then continue to work with that file. So let's make use of this and see some further changes to our git repository.

Writing the checkout command automatically stages the data to commit.

git reset Head "some filename " read about this. This will basically unstage a staged file.

To add these repositories online,

git remote add origin <repository URL> adds the remote online repository

git push -u origin master pushes the local git repository to the origin to the master branch

Why JS frameworks?

Complicated manipulations with DOM

There's a standard set of functionalities which can be coupled in a software library

Read difference between software library and framework

library is basically a collection of functions.

framework is a particular way of implementing the application at hand

When using a framework, it will always take charge of the main code of the web application.

When using a library however, your vanilla js code is the main code with additional functions provided by the library.

Imperative vs Declarative programming

Imperative programming means that the developer mentions what is to be done along with how exactly should it be done.

Declarative programming means that the developer only mentions what is to be done and leaves the actual implementation to the react framework.

usually frameworks use the declarative approach.

Three Ss of react are speed, simplicity and scalability.

just type npm start and your React application will start up, and be served using that built-in server that is part of the create-react-app ecosystem.

Much of the HTML code that you see in a react app can be called as elements. It is a collection of these elements which collectively make a component.

Any css class in react will have the name ClassName and not just class as in normal HTML.

This is because writing just class in the inline css will be confused with actual classes used in the js code elsewhere. This is the reason why we just use ClassName instead.

Rendering The View to the DOM

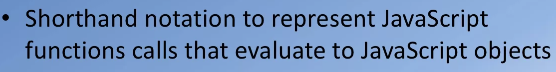
Inside src folder, inside index.js file, we have ReactDOM.render(<App />, document.getElementById('root')

what this basically says is that "Render the content returned from the App component and paste it in the space on the webpage pointed by the id root "

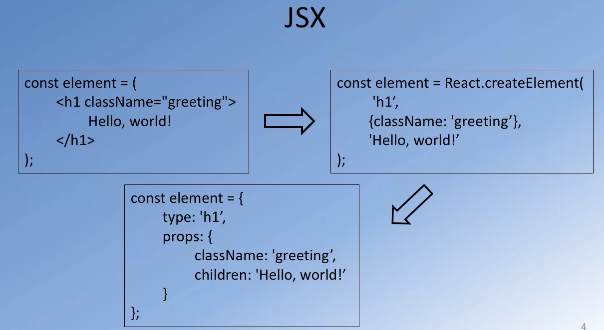
What we basically do is we build the component in the app.js folder inside of the src folder. Then we import it inside of index.js in the same directory. Finally, the view is rendered inside the index.js file and pasted on to the empty space pointed by id root in index.html in public folder.

**JSX**

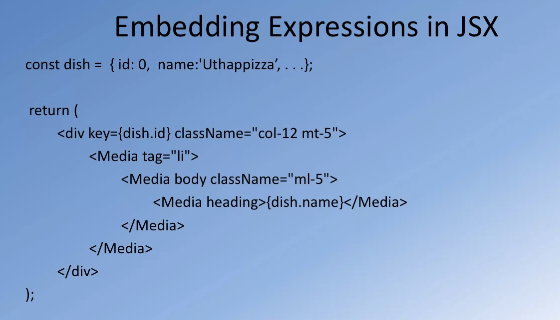
Syntactic Extension to Javascript



you obviously notice immediately some HTML like syntax in the code, and some specification of attributes for the HTML like elements and so on. But, this actually gets mapped in react into a corresponding JavaScript object here. So as you can see, this is mapped into a react.createElement object. This call to the react.createElement with the three parameters as shown here, will result in a JavaScript object being created finally.



Many of the standard HTML tags have corresponding react components that are already defined, predefined react components. And hence, we are able to use them freely within our react code. Not only that, there are fact that you are mixing UI logic with rendering logic means that you should be able to easily embed JavaScript like expressions into your JSX code. And indeed that is quite possible with JSX. So, for example, you could easily declare a JavaScript object like this here, and then correspondingly, you can incorporate their properties from this JavaScript object into the code that you render here. So as you can see here, for a div, I'm giving a key attribute as dish.id, which has actually derived from the JavaScript object there. And then correspondingly, you can see the use of the media heading. And then in there written braces, you'll see dish.name being declared here, and so on. And so, this code shows you how JavaScript expressions can be embedded into JSX code, as and when required. Now, we will encounter more of such examples as we double up our react application later, and then at that point, you will see how versatile JSX is for specifying various react element.



for a div, I'm giving a key attribute as dish.id, which has actually derived from the JavaScript object there. And then correspondingly, you can see the use of the media heading. And then in their written braces, you'll see dish.name being declared here, and so on. This code shows you how JavaScript expressions can be embedded into JSX code, as and when required.

Now this code, I think is the crux of how states are used

The js part of bootstrap cannot be directly used together with react and so instead we use ReactStrap.

**COMPONENTS**

We can look at a component as a unit that returns a group of React elements that together render a part of the screen. So the components acts as a unit for gathering together a bunch of React elements with a common purpose. Now the use of react components enables us to split the user interface into multiple independent, reusable pieces. So, when you define a component, which renders a particular kind of user interface part, then that component can easily be reused in your UI wherever you require similar kind of behaviour or similar kind of rendering of the part of the user interface. So, components enable us to break down the entire UI in to smaller reusable pieces. Furthermore, you can easily control what a component renders by supplying inputs to the component.

when you define your own components and add them to React, you should always start the name of the component with a capital letter. This way the React compiler recognizes that that should be

Start transcript at 2 minutes 26 seconds2:26

mapped into a corresponding React.createElement. So as you re-explore GSX in the previous lesson, we saw how the component defined in HTML syntax is mapped into a React.createElement function call. And tags that usually start with a small letter, or lower case letter, is typically integrated as a DOM tag.

**State**

The state stores in it, Properties related to this component that we can make use of.

each component can store its own local information in its state

 this information is private and fully controlled within that component.

Now state within a component can easily be passed to

its children through the use of props

If we need to store the state, we specifically need to use the class based component. If no state is required then a functional component will do.

This is because only class component can have state. This in turn is because the state data is stored inside a constructor which can only exist in a class.

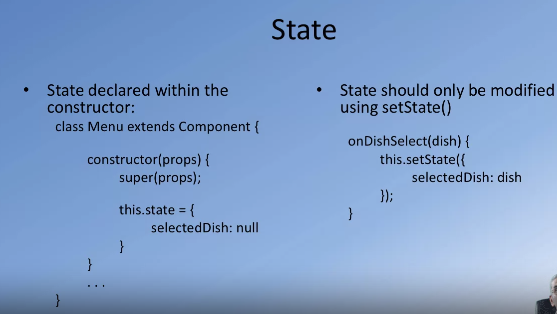
you cannot directly go and update the state by changing the property values.

Instead, any update to a state of the component has to be

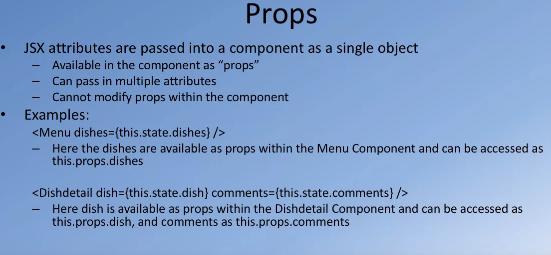
done through the use of that Sect State method.

So, whenever you need to make any changes to the state of the component,

you always say this.setstate and then supply the property that you want to modify







When you make use of the time component in your own component,

then whatever attributes that you specify to

the JSX element will be passed in as props to the child component.

So these attributes will be available as props within

the child component and you can

easily pass in multiple attributes to the child component.

Let's take a look at an example.

So in these examples,

you can see in the first case,

we are using this menu component within

your current component and then you're passing in dishes as one of the attributes.

Now when you pass this in,

then within the menu component,

this dishes information that your passing will be available as

this.prop.dishes and can be used within the menu component for it to render its view.

Similarly, you can pass in multiple attributes.

If you choose to for a child component and

the child component will get access to each of these multiple attributes, as this.

props.image.

So, as you see this.props.dish or this.props.

comments and so on.

So, this is an easy way of a parent component passing information to its child component.

So in this case,

a parent component maybe storing the state and

then the state information can be passed to the child component through

the use of the props in the form of attributes that you specify to the JSX statements.

Very often, your react application will be implemented as a hierarchy of components.

So in that case,

you may wish to lift the state up to an ancestor of the current component.

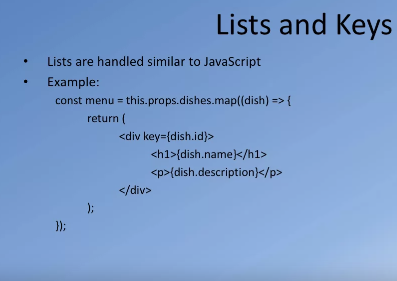
So for example, if several components have a shared state and

any change to the state needs to be automatically seen by all these components,

then it is better to move the state to one of the ancestors,

common ancestor of all these components,

and ensure that any state changes will be sent back up to that common ancestor



for each element of the array I can then laid out as a list of item.

When you do list items in react.

For each of the items it is important to specify

the keys for each element inside an array.

So as you can see in this example,

I am using an outer div array a very specific keys is equal to dish ID.

So in this case since each dish will have its own unique ID,

I'm using the dish ID as the key when I lay out the list of items data.

The reason to specify a key in react,

is that when there are changes to

the number of items in the list maybe

a new item is added into the list or an item is removed from the list,

any such changes to the data may cause re-rendering of the list items.

So, when react is doing that,

it will use these keys to identify which items have been removed and so

appropriately partial re-rendering of this information is facilitated in react.

As you recall, react only modifies those parts of the DOM tree that need to be updated.

As we realize, a React application is made up of multiple React components that

are connected together to form the entire screen of your React applications view.

Now, each React component as and when it is required will be created by React

and then added into the DOM of your entire application.

So, every time a React component needs to be included into your applications view,

then the component that hosts the specific part of the view will be

created and added into the overall React component hierarchy.

So, a component passes through what we call as the life cycle for the component.

Now initially, the component doesn't exist,

then the component gets created and then once it is created,

then it can be mounted into your React application

at an appropriate point in the hierarchy of components,

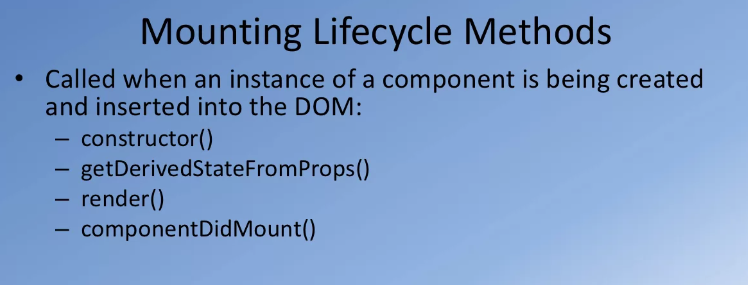
then it will exist at that point for a period of time and then subsequently,

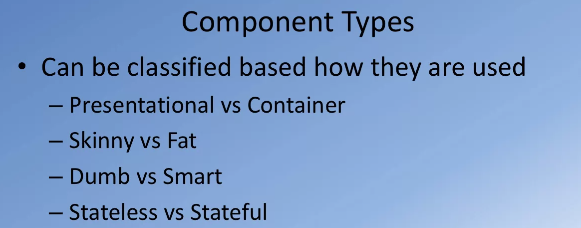
maybe you no longer require that view,

and so at the point,

the component can be removed from this hierarchy.

So that will be an unmounting of the component.





we can talk about presentational components that are purely responsible

for rendering the view based on the props that they are passed in,

and then you could have container components that simply make

use of presentational components and pass props to

them and they are responsible for storing the state (the container components).

Similarly, we can talk about these as skinny and fat components.

Skinny components mean purely responsible for rendering the view and

fat components having a lot more information being tracked there in the form of state.

Similarly, we can even label them as dumb and smart components,

stateless and stateful components.

No matter how you classify them,

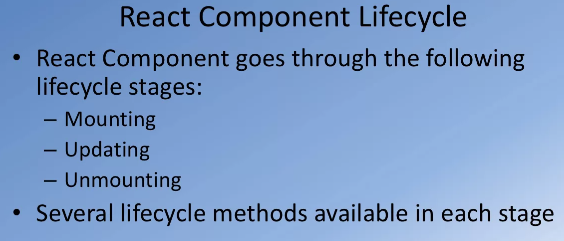
you begin to see that they can view components as of two kinds.

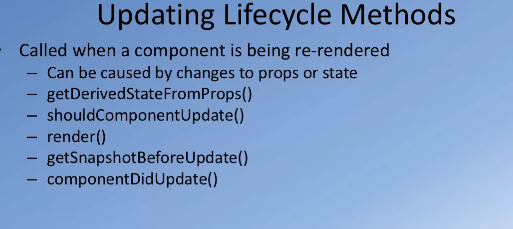
Again, these are informal ways of classifying the components.

There is no formal description of these kinds of components in React itself,

but this helps us to structure our application when we view

the components of different kinds as delineated here.





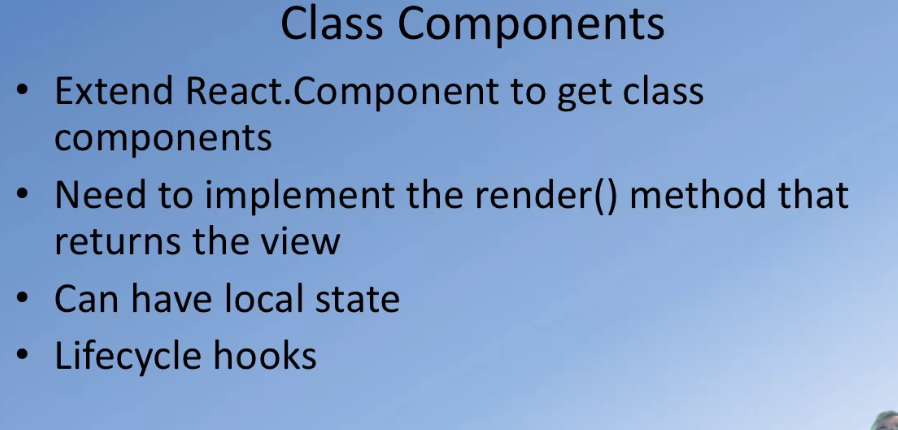
for components that simply

work only using their props that are sent by their parent,

there is an even simpler way of

implementing the components what are called as Functional Components.

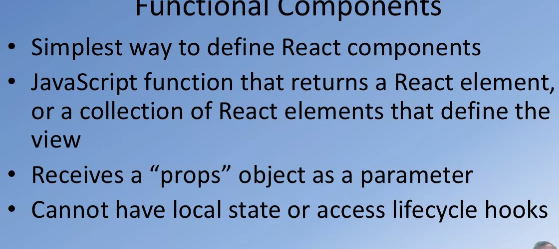
Then we implement a simple function that returns a bunch of react elements

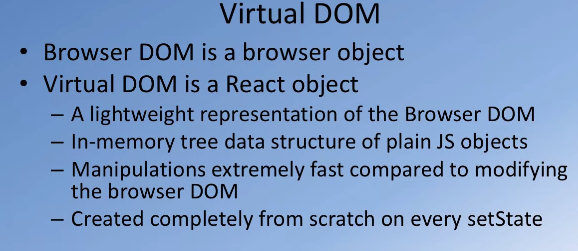


But if your component is a very simple component and

doesn't require this much of elaborate implementation,

there's a much simpler way of implementing React Components





whenever you call setState,

you are changing the state with a new React application,

which also means that it is possible

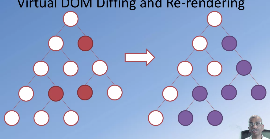
that some of the props passed into the child components to change,

and so the child components may have to be re-rendered.

So, in the Virtual DOM hierarchy that you build up in your React application,

not all components make it affected by changes to the state of your application.

So, as an example,



on the left side,

I show you a Virtual DOM tree here.

And in that Virtual DOM,

you see I have marked three components there with red red color.

These three components are the ones that have been affected by

a change in the state for my React application.

Now, what that means is that you will have to re-render your view in the browser,

that means that you will need to manipulate the Browser DOM

to make the Browser DOM correspond to

the changes that have been effective in the Virtual DOM.

So, the React application's work is that they will do

a difference between the previous version of

the DOM and the current modified version of the DOM.

So, to notice which parts of the Virtual DOM tree have changed,

then you know which of the components need to be re-rendered.

So in this case, for example,

when these three red components are effected due to the change in the state,

then it may result in re-rendering of

all those nodes that are marked in purple to the right of the screen.

So, only that part of the DOM tree may have to be re-rendered to

ensure that the changes are reflected into the Browser DOM.

So how is this actually done?

Now, with React because React maintains the Virtual DOM which is a React object,

and as we have understood,

manipulating the Virtual DOM is very very fast and effective.

So every time there are changes to the Virtual DOM

and you are at the point of re-rendering the view in the Browser DOM,

then to update the DOM,

React runs in diffing algorithm.

The diffing algorithm will detect all the nodes that have changed.

And so, once it detects all the nodes that are changed as we have seen,

the red color nodes that were marked in the tree there,

this may result in updates to

the entire sub-tree if the diffing detects that two elements are of different types.

So, the diffing algorithm identifies the minimum number

of components or minimum part of the tree that needs to be

updated in order to make the modified version in sync with that Browser DOM.

Then you are rendering multiple items,

for example, in a list.

You can use the key attribute in

the list items in order to indicate which child elements are stable.

Indeed, that is the reason when we rendered list,

we always supply the key attribute to each list item,

and each key attribute was a unique identifier for that particular item in the list.

So, when the diffing algorithm works,

if it notices that some parts of the list cannot change,

they don't need to be re-rendered.

And so, it'll re-render only those list items that have actually been modified.

Now, with the React 16,

there is a new version of the React diffing algorithm called React Fiber.

This is a new reconciliation algorithm that has been launched with

React 16 and it is a lot more faster in performing

the diffing and then identifying what needs to

be changed in the Browser DOM to update the views.

With this quick understanding of what the React Virtual DOM is,

let's move on to learn more about how to make use of the React router in this lesson.

whenever

**THIS IS WHY WE ALWAYS USED THE KEY ATTRIBUTE ALONG WITH LISTS WHEREVER WE USED LISTS. IT BECOMES EASIER FOR REACT TO RUN THIS DIFFING ALGORITHM WHEN IT HAS KEYS TO COMPARE EACH ELEMENT WITH.**

A typical React Application may consist of multiple quote unquote pages.

Meaning that you have multiple views and you want a way of navigating among the views.

Then, we develop any website for example.

A website canters of multiple pages and we provide the navigation tool,

the use of a navigation bar or links to be able

to go to the different pages of our website.

Now, within a web application also,

you might have multiple pages and you want to be able to

navigate among the views that are rendered by these various pages.

So, this is where the React Router comes to our aid.

The React Router module,

we install it into our React Application and then take the help of the various features

provided by the React Router module to enable us

to navigate among the views of our application.

**React router parameters can be used to pass data between different routes in our applications.**

**React router passes 3 props as parameters match object, location object, location history.**