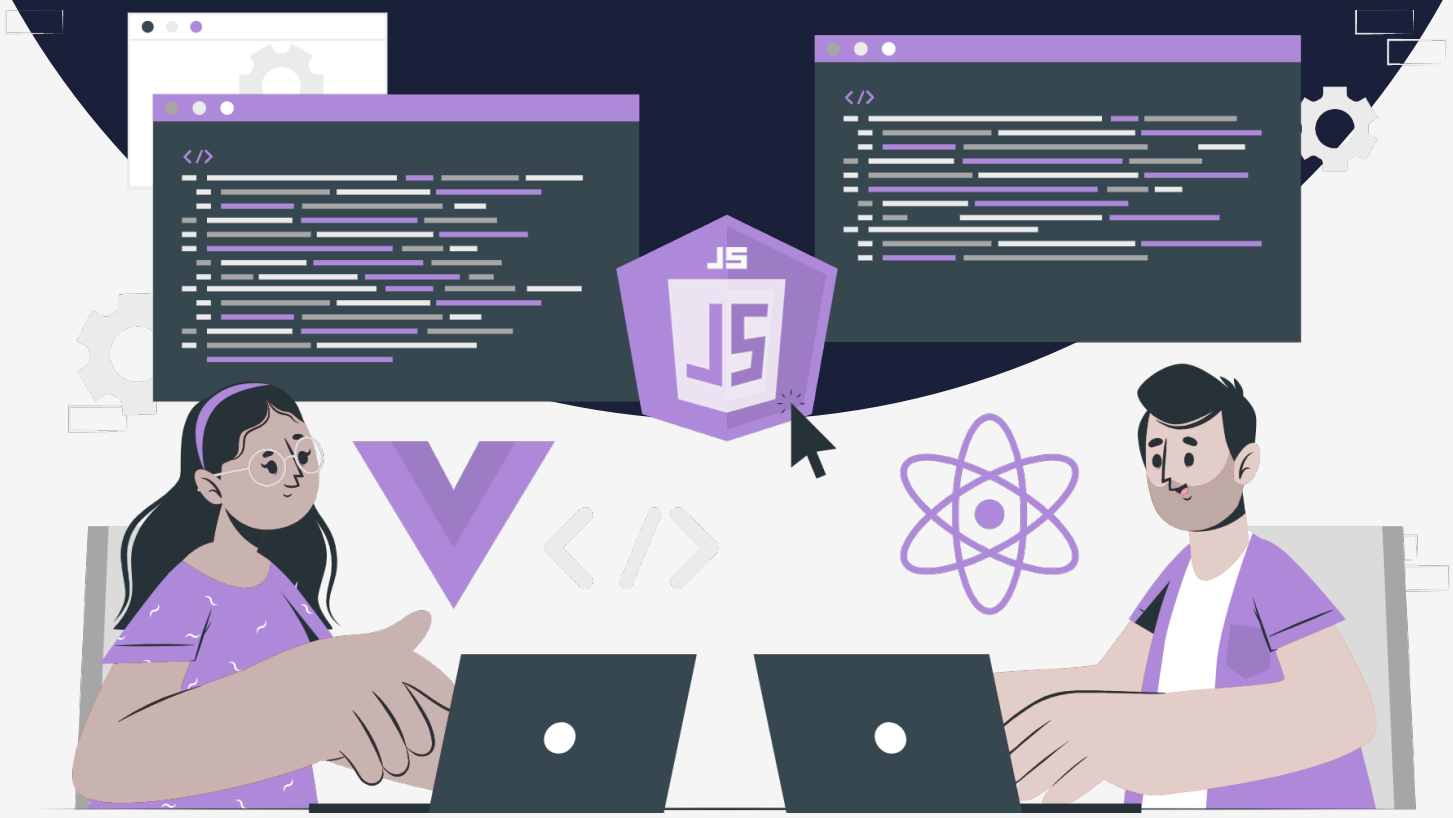
Lesson:

How to use .map() to render array & Optimizing lists with key arguments



PW SKILLS

Topics Covered:

1. Rendering multiple components with list
2. What are React Keys?
3. Why does React need Keys?
4. Diffing algorithm.
5. Conclusion.

Rendering multiple components with list
Let's take an example of how to render multiple components.

```javascript
const People = [
{
 id:0,
 name:'Pradeep',
 profession:'student',
},

{
 id:1,
 name:'Manish',
 profession:'Teacher',
},

{
 id:2,
 name:'Ameya',
 profession:'Teacher',
},
{
id:3,
 name:'Qamar',
 profession:'Teacher',
},
{
id:4,
```

```
 name:'Prithvi',
 profession:'student',
},
{
id:5,
 name:'Vineet',
 profession:'Teacher',
},
]
```

Let's say we want to only show the list whose profession is 'Teacher'.We can use filter() method to return those people.

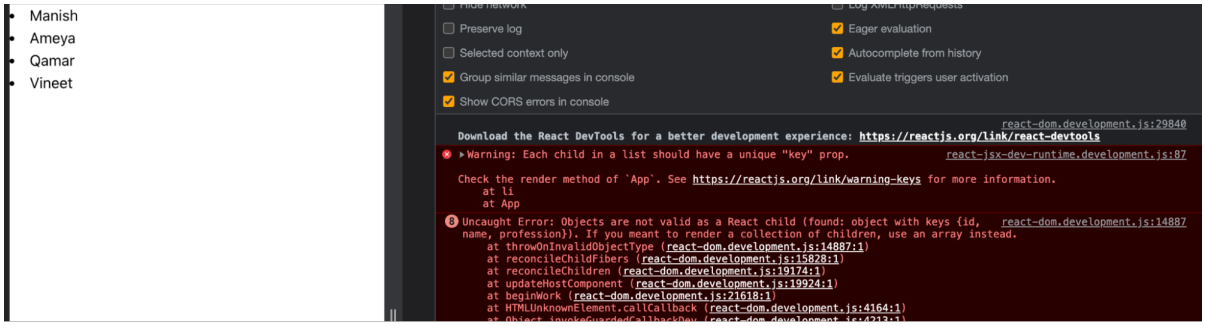
Create a new array of 'teachers'

```
JavaScript
const teachers = people.filter(person=> person.profession === 'Teacher');
```

Now map over "teachers"

```
JavaScript
const listitems = teachers.map(person =>
   <li>
     {person.name}
   </li>)
 return <div>
   {listitems}
 </div>;
```

Output:



We get the output but there is some warning in the console i.e **“Each child in a list should have a unique"key " prop".**

What are React Keys?

Simply they are props that are passed in child elements of lists in order to:

- identify which elements are added.
- identify which elements are updated.
- identify which elements are removed.

Hence, keys serve as identification for an element just like how passports are used to identify people.It must be a string or a number that uniquely identifies it among other items in that array.

However it is used to identify which items have changed, updated or deleted from the lists.It becomes more useful when we dynamically create components or when we alter the list.

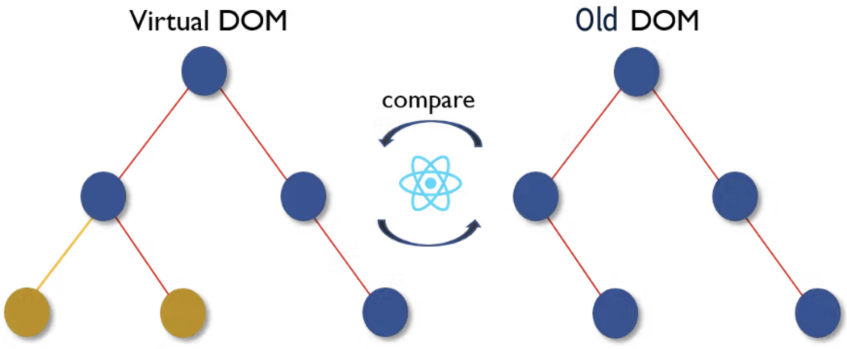
Why does React need Keys?

Imagine We have some files in our computer that do not have any names. We remember them by their order names like- the first file, the second file and so on. But once we delete a file , It would get confusing.File names in a folder and JSX keys in an array serve a similar purpose. They let us uniquely identify an item between its siblings.

Because of React's **Diffing algorithm**, We can identify elements by their Id, names, index,className etc.

Diffing algorithm

React is made up of a tree of components. Whenever there is any change in the component, React re-renders its component into virtual DOM.The Diffing algorithm checks if there is any change in the component, it compares the new virtual DOM with the old DOM at each level of the component tree**.**



The algorithm finds the minimum number of operations required to update the real DOM. This is how it does it:

1.Compare node by types(vs)

If there is any difference, it will be destroyed and built from scratch.

```
JavaScript
// virtual DOM
<div><MyComponent/></div>

// real DOM
<span><MyComponent/></span>
```

2.If nodes have the same type, compare by attributes.

If different, only update the attributes

```
JavaScript
// virtual DOM
<div className="btn-info" title="name" />


// real DOM
<div className="btn" title="name" />
```

This results in an update of className to “btn-info”.

For lists, React will recurse on both their children simultaneously, find any differences, then patch them to the real DOM if there are any.

```
JavaScript
//Virtual DOM

<ul>
  <li>one</li>
  <li>Two</li>
  <li>Three</li>
</ul>

//Real DOM

<ul>
   <li>One </li>
   <li>Two </li>
</ul>
```

This results in the Three being added after Two.

But now, instead of adding an element at the bottom of the list, We want to add elements at the beginning of the element.

```
JavaScript
//Virtual DOM
<ul>
  <li>Zero</li>
```

```
   <li>One</li>
   <li>Two</li>
 </ul>

 //Real DOM

 <ul>
    <li>One</li>
    <li>Two</li>
 </ul>
```

React will re-render every single <li> to the real DOM because it doesn't realise that it can simply add zero to the beginning of the list.

This inefficiency can cause problems, especially in larger apps. Hence, keys provide a simple solution to this issue.

We need to give each array item a key .

```
JavaScript

<li key={person.id}>...</li>
```

Note:

- **Arrow functions implicitly return the expression right after "=>" , so we don't need a return statement.**

```
JavaScript
 const listitems = teachers.map(person =>
    <li>.....</li> //implicit return
);
```

- **However we must write return explicitly if our => is followed by a curly bracket `{}`**

```javascript
const listitems = teachers.map(person =>{
  return
   <li>.....</li>
});
```

Conclusion:

- The main purpose of keys is to help React differentiate and distinguish elements from each other, increasing its performance when diffing between the virtual and real DOM. To use keys, simply add the “key” prop inside an element such as <li>.
- Unique ids is the best value to assign to keys. You can only use index as the key if the list is static (and list elements will not be updated or removed) and the elements in it have no unique identifier.