MERN Stack Assignment 1

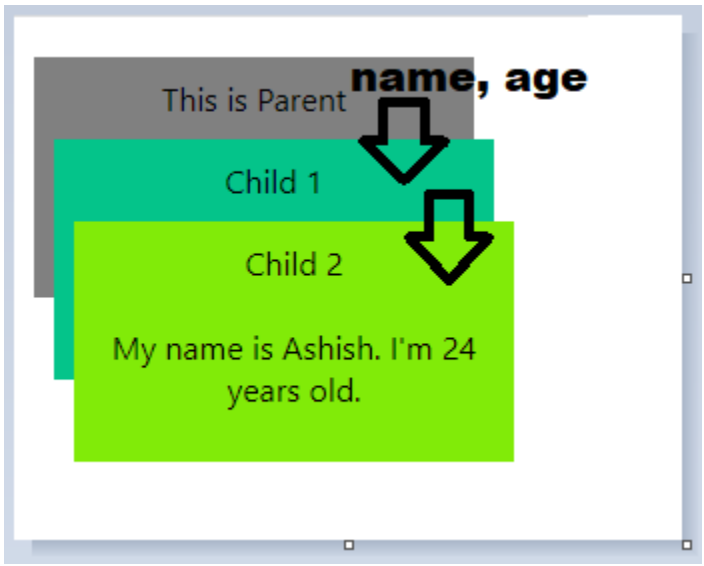
1. What exactly do you mean when you say "prop drilling," and how do you avoid it?

Ans.

Props are basically the parameters. In React props can be used to pass data between the components.

Props Drilling refers to a process of passing props from a higher-level component to a lower-level component.

Suppose there is a requirement of data at the lower level, then in this situation the same data is being sent at almost every level through props. Below image describes prop drilling.



Here name and age are the data which was required at the lower level so, we had to send this data through props from top level to lower level. This is the prop drilling.

The problem with the Prop Drilling is that whenever any component requires the data from the Parent Component, the data will have to be sent through props coming from each level, regardless of the fact that data is not needed by any components in the between but only needed by the last one.

Prop Drilling can be avoided with the use of React's useContext hook.

- React's useContext hook makes it easy to pass data throughout the app without manually passing props down the tree.
- The useContext hook is based on Context API and works on the mechanism of Provider and Consumer.
- Provider can be used to wrap the components and pass the values that the child component may expect.
- Consumer in Context refers to someone who consumes the values. It can retrieve the values that were passed to the same Provider using the useContext hook.

App.js

```
import "./App.css";
import Parent from "./Parent";

function App() {
  return (
    <div className="App">
      <Parent />
    </div>
  );
}

export default App;

```

Parent.js

```
import React, { useState } from "react";
import Child1 from "./Child1";
import "./Parent.css";

var Context = React.createContext(null);

function Parent() {
  const [name, setName] = useState("Ashish");
  const [age, setAge] = useState(24);     'set
  return (
    <Context.Provider value={{ name, age }}>
      <div className="Parent">
        This is Parent
        <Child1 />
      </div>
    </Context.Provider>
  );
}

export default Parent;
export { Context };
```

Here Provider is used to wrap the components and pass the values that the child component expects.

Child1.js

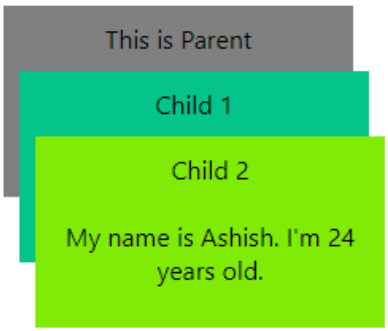
```jsx
import React from "react";
import "./Child1.css";
import Child2 from "./Child2";

function Child1() {
  return (
    <div className="Child1">
      Child 1
      <Child2 />
    </div>
  );
}

export default Child1;
```

Child2.js

```jsx
import React, { useContext } from "react";
import { Context } from "./Parent";
import "./Child2.css";

function Child2() {
  const { name, age } = useContext(Context);
  return (
    <div className="Child2">
      Child 2
      <br />
      <br />
      {`My name is ${name}. I'm ${age} years old.`}
    </div>
  );
}

export default Child2;
```

Here Child 2 will retrieve the values that were passed to the same Provider using the useContext hook.

Output



2. In React JS, how do you add validation to props?

Ans.

Props validation is a tool that helps developers to avoid unexpected bugs and problems. React Components used a special property PropTypes that helps to catch bugs by validating data types of values passed through props.

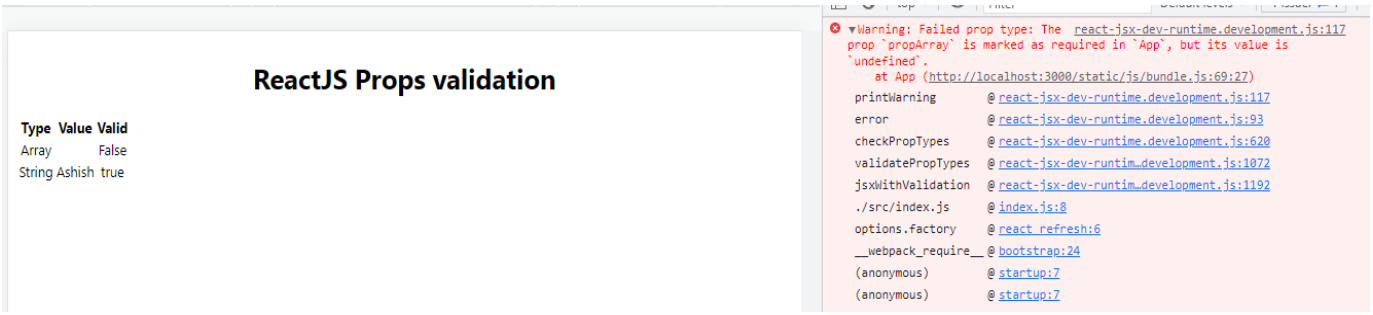
Component.propTypes *is used for props validation in react component* .

When some props are passed with an invalid type, a warning on the Javascript console is received.

We can provide default props value after specifying the validation with App.defaultProps.

```
import "./App.css";
import PropTypes from "prop-types";

function App(props) {
  return (
    <div className="App">
      <h1>ReactJS Props validation</h1>
      <table>
        <tr>
          <th>Type</th>
          <th>Value</th>
          <th>Valid</th>
        </tr>
        <tr>
          <td>Array</td>
          <td>{props.propArray}</td>
          <td>{props.propArray ? "true" : "False"}</td>
        </tr>
        <tr>
          <td>String</td>
          <td>{props.propString}</td>
          <td>{props.propString ? "true" : "False"}</td>
        </tr>
      </table>
    </div>
  );
}

App.propTypes = {
  propArray: PropTypes.array.isRequired,
  propString: PropTypes.string,
};
App.defaultProps = {
  // propArray: [1, 2, 3, 4, 5],
  propString: "Ashish",
};

export default App;
```

When data is not sent to the array props which is validated as required, warning is shown in the developer console.



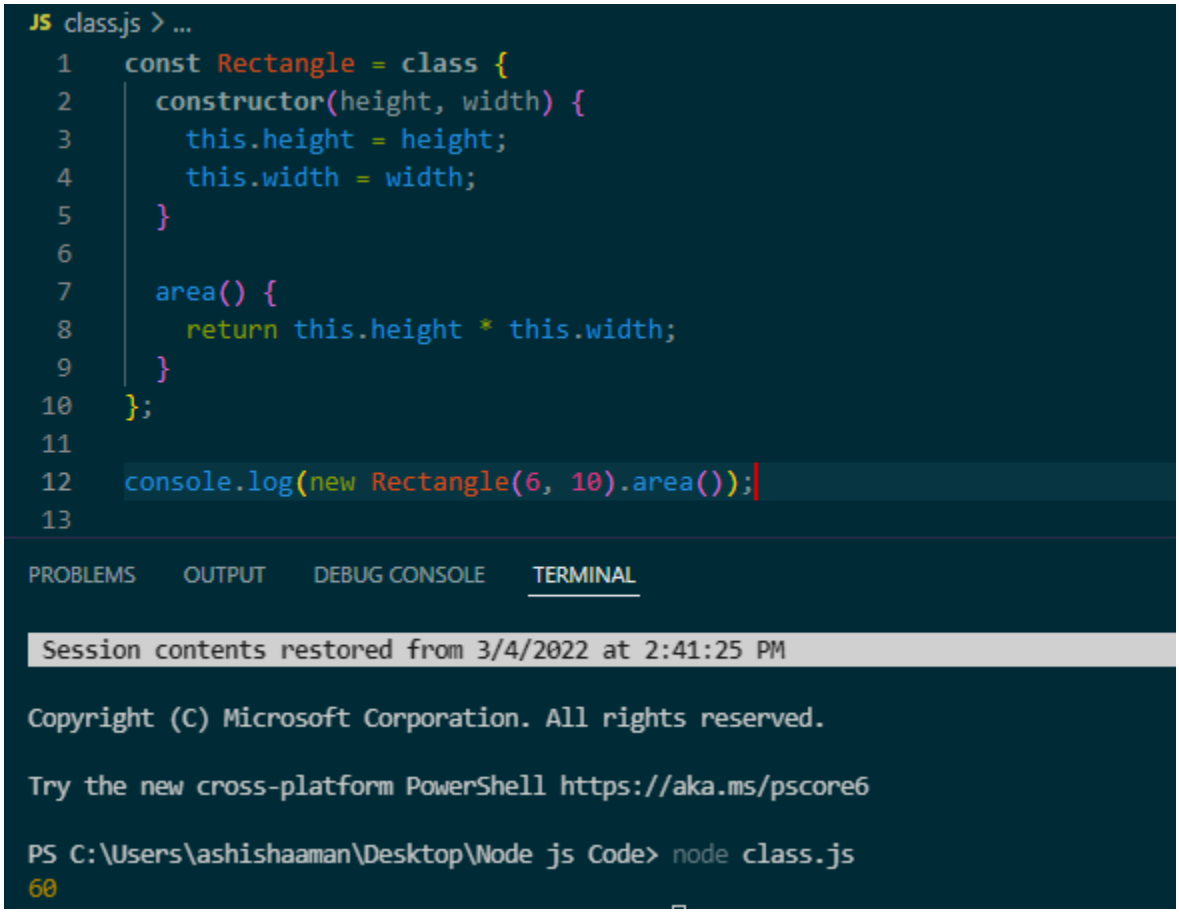
3. Is it possible to use classes in NodeJS?

Ans.

Yes, it's possible to use classes in NodeJS.

The body of the class is within the curly brackets {} and inside the body we define the class members. Class body can have methods and constructors. The constructor is a special method which is used for initializing the object created with the class.

Below example is of a class Rectangle in Node js used to find the area



4. What is the purpose of super(props)?

Ans.

A child class constructor cannot make use of this reference until super() method has been called. The same applies for ES6 sub-classes as well. The main reason for passing props parameter to super() call is to access **this.props** in the child constructors.

App.js

propdrilling > src > JS App.js > App

```
1   import "./App.css";
2   import Demo from "./Demo";
3
4   function App() {
5     return (
6       <div className="App">
7         <Demo name="Ashish" />
8       </div>
9     );
10  }
11
12  export default App;
13
```

Here prop is passed to the child component.

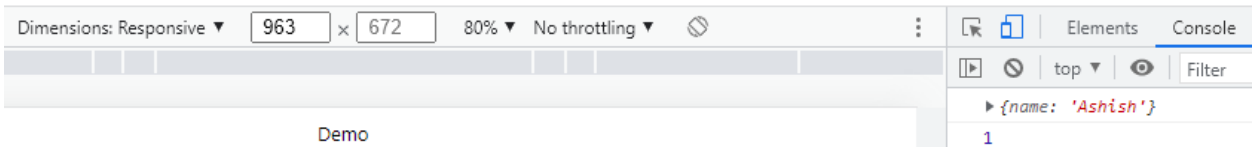
Demo.js

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

export default class Demo extends Component {
  constructor(props) {
    super();
    console.log(this.props);
  }
  render() {
    return <div>Demo</div>;
  }
}
```

If props is not used with super() → we won't be able to use this.props inside the child constructor. The above example returns undefined in the console.

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

export default class Demo extends Component {
  constructor(props) {
    super(props);
    console.log(this.props);
  }
  render() {
    return <div>Demo</div>;
  }
}
```

When we use super(props) we are able to use this.props within the child constructor.



5. Why are the Express app and server separated?

Ans.

The Express App and Server are kept separated because the separation of the application logic from the server allows the code to be modular and follow a MVC (Model-View-Controller) model. The separation is essential to reduce coupling and to encapsulate and abstract the inside logic of application.